

国际信息工程先进技术译丛



Springer

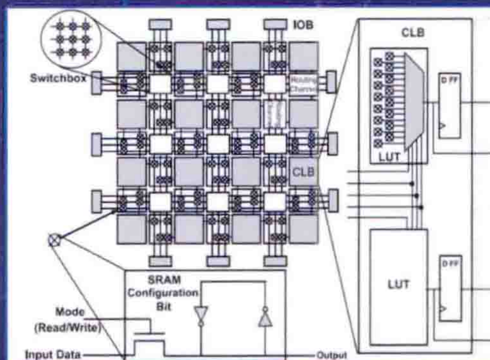
FPGA安全性设计指南

Handbook of FPGA Design Security

（美） Ted Huffmire
Cynthia Irvine
Thuy D.Nguyen
Timothy Levin
Ryan Kastner
Timothy Sherwood 著

房亮 吴少俊 闫蕾 宫永生 译
夏宇闻 审






机械工业出版社
CHINA MACHINE PRESS

国际信息工程先进技术译丛

FPGA 安全性设计指南

Ted Huffmire
Cynthia Irvine
（美）Thuy D. Nguyen 著
Timothy Levin
Ryan Kastner
Timothy Sherwood
房 亮 吴少俊 闫 蕾 宫永生 译
夏宇闻 审

机 械 工 业 出 版 社

图书在版编目（CIP）数据

FPGA 安全性设计指南/（美）霍夫曼（Huffmire，T.）等著；房亮等译. —北京：机械工业出版社，2014. 2
（国际信息工程先进技术译丛）
书名原文：Handbook of FPGA design security
ISBN 978 - 7 - 111 - 45783 - 1

Ⅰ. ①F… Ⅱ. ①霍…②房… Ⅲ. ①可编程序逻辑器件 - 系统安全性 - 系统设计 - 指南 Ⅳ. ①TP332. 1 - 62

中国版本图书馆 CIP 数据核字（2014）第 025370 号

机械工业出版社（北京市百万庄大街 22 号 邮政编码 100037）
策划编辑：林春泉 责任编辑：任 鑫
版式设计：霍永明 责任校对：李锦莉
责任印制：刘 岚
北京京丰印刷厂印刷
2014 年 5 月第 1 版 · 第 1 次印刷
169mm × 239mm · 11 印张 · 209 千字
0 001—3 000 册
标准书号：ISBN 978 - 7 - 111 - 45783 - 1
定价：58. 00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务	网络服务
社服务中心：（010）88361066	教材网：http：//www. cmpedu. com
销售一部：（010）68326294	机工官网：http：//www. cmpbook. com
销售二部：（010）88379649	机工官博：http：//weibo. com/cmp1952
读者购书热线：（010）88379203	**封面无防伪标均为盗版**

现场可编程门阵列（FPGA）已经成为嵌入式系统设计的主要应用技术之一。可重构器件由于融合了硬件和软件的特性，能够在专用集成电路的高性能和CPU的可编程性之间找到自己的应用空间，产生更好的应用效果。与此同时，FPGA安全性设计的问题也日益突出。目前，关于FPGA安全性设计的专门著述较少，FPGA设计者很难针对具体应用进行系统的安全性分析和设计。

本书通过理论阐述并结合实用设计的方式，通过举例说明FPGA安全性设计的问题如何进行解决。作者从如何编写顶层设计的形式化说明开始，逐步涉及低层硬件电路的各项强化机制，并分为多个层面对FPGA安全性问题和解决方案进行了全面的阐述。作者结合近年来在计算机安全性理论、编程语言、编译器和硬件设计等领域中的最新进展，与FPGA设计中的安全性问题作为一个整体予以阐述，创建了一整套静态和动态分析互相配合的多样化设计技术，使得使用商业芯片构建的FPGA系统有可能成为一个稳定、可靠和安全的强健系统。

本书旨在为EDA（电子设计自动化）和FPGA领域工作的研究者和实践者们提供一整套FPGA安全性设计的管理实用方法。本书适合在公司、工厂和政府研究实验室工作，从事FPGA设计的工程师和学术界人士阅读。尤其是对FPGA安全性要求较高的领域。同时也适合致力于FPGA安全性设计研究的人士，用以提高专业技能。

Translation from English language edition:

Handbook of FPGA Design Security

by Ted Huffmire Cynthia Irvine Thuy D. Nguyen Timothy Levin Ryan Kastner and Timothy Sherwood

北京市版权局著作权合同登记图字：01-2012-1048号

译 者 序

随着航空、航天等军工领域嵌入式数字系统应用的多元化，可编程逻辑设计技术由于其本身的综合成本优势和灵活性，已经成为核心的支撑技术之一，并在高可靠领域有了大规模的应用。与此同时，如何解决可编程逻辑的安全性设计问题也成为了一项迫切需要进行研究的工作。

本书是针对FPGA设计安全保障的指南专著，泰德·霍夫曼，辛西亚尔·伊凡等六位作者是FPGA设计安全领域的资深专家，对FPGA安全性防护有全面深入的研究和独到的见解。本书不仅对FPGA设计中可能出现的安全隐患问题进行了全面、细致的分析，并且提出了清晰的解决思路，对相关领域的科研及工程技术人员具有很强的指导意义和借鉴价值。

当夏宇闻教授将本书的英文版《Handbook of FPGA Design Security》推荐给我们时，我们欣喜地发现本书对工程中出现的FPGA安全性设计问题给予了系统的分析。当夏宇闻教授建议我们结合工程应用中的实践经验，将本书翻译成中文时，我们立即欣然答应。我们所在的单位是中国科学院空间应用工程与技术中心，主要负责载人航天工程空间应用系统的工程组织管理及总体技术研究工作，成立以来承担了近300台（套）有效载荷的工程研制工作，积累了丰富的可编程逻辑设计经验。我们相信本书中文译稿的出版将对国内的FPGA安全性设计工作产生积极的影响。

参与本书翻译工作的主要译者是从事载人航天工程空间应用系统中嵌入式系统多年研究的技术专家，他们是中国科学院空间应用工程与技术中心的房亮副研究员、吴少俊副研究员，闫蕾副研究员和宫永生工程师，均为载人航天工程空间应用系统相关型号任务的负责人和核心骨干。本书由夏宇闻教授负责审校。

在《Handbook of FPGA Design Security》中文版即将出版之时，特别向中国科学院空间应用工程与技术中心领导和同事对我们工作的支持、向北京航空航天大学夏宇闻教授和机械工业出版社林春泉编审的悉心指导和帮助表示衷心的感谢！

由于FPGA安全性设计技术在国内的相关研究较少，译书中难免有不当之处，敬请读者批评指正。

译者

2014年4月

审校者序

在嵌入式军用数字系统设计领域，FPGA 有着巨大的发展潜力，然而也存在着许多安全隐患。近年来，在推广 Verilog 设计方法学的同时，我也一直在关注着 FPGA 安全性设计方面的文献和书籍。

2011 年底，在中科院空间应用工程与技术中心工作的、我的前硕士生吴少俊向我咨询 FPGA 安全性设计问题时，我立即推荐了我刚读过的一本由 Springer 出版社 2010 年底出版的新书《Handbook of FPGA Design Security》。

该书由泰德·霍夫曼（Ted Huffmire），辛西亚尔·伊凡（Cynthia Irvine）等六位具有丰富实践工作经验的教授、专家共同编写。他们是美国著名的海军研究生院和加州大学计算机科学系 FPGA 设计安全领域的资深专家。该书源于他们在 FPGA 设计安全保障领域研究中多年积累的宝贵经验的总结。书中的内容十分广泛，几乎涵盖了 FPGA 设计安全保障方面的所有问题，该书阐述了造成 FPGA 设计安全隐患的主要原因，并且提出了解决安全隐患问题的清晰思路。我认为该书所介绍的安全保障思路对中国国防领域的 FPGA 设计者有很大的参考价值。

我希望吴少俊和他的同事们能够认真读一读该书，并花点时间把它翻译成通俗易懂的中文。有了好的中文书籍，一定会促进国内 FPGA 设计安全保障工作能更快更健康地开展。所以我答应担任该书译稿的审校。吴少俊征求所在部门意见后同意了我的建议，不久我就把该书和吴少俊介绍给了机械工业出版社的林春泉编审，出版社很快向 Springer 出版社购买了该书的中文版版权，随后签订了翻译合同。

在一年半的翻译过程中，参与翻译工作的人员非常认真，经常通过电子邮件与我切磋难以用通畅中文表达的章节段落，译稿曾经多次反复修改。由于水平和时间有限，我的审校不免存在一些遗漏和错误，敬请细心的读者不吝指教。

在本书出版之际，让我衷心地感谢曾经为本书出版做出过贡献的所有同仁。

夏宇闻
北京航空航天大学电子信息工程学院退休教授
北京至芯科技公司 FPGA 设计培训顾问
2013 年 9 月 18 日

原书前言

本书旨在为EDA（电子设计自动化）和FPGA领域工作的研究者和实践者们提供一整套FPGA设计安全性管理的实用方法。本书的读者群包括公司、工厂、政府研究实验室、相关领域的工程师和学术界人士。

本书将理论基础的阐述与实用设计方法的介绍紧密结合，通过举例说明了FPGA设计的安全性问题是如何解决的。为了透彻理解FPGA系统在运行中的风险和生命周期等一系列问题，本书从如何编写顶层设计的形式化说明书开始，一直讲到低层硬件电路的强化机制，分多个层面对FPGA安全性问题和解决方案进行了全面的阐述。在这一过程中，作者将近年来在计算机安全性理论、语言、编译器和硬件等领域中的进展，与FPGA设计中安全性的考虑作为一个整体予以阐述，从而创建了一整套静态和动态分析互相配合的多样化设计技术，使得用商业化芯片构建的FPGA系统有可能成为一个稳定、可靠、经得起考验的强健系统。

在本书出版之际，我们要感谢那些曾经给我们提供帮助的人们，他们的支持是这本可重构硬件安全性著作之所以能取得成功的关键。我们要特别感谢路易斯安那州立理工大学的Andrei Paun和Jason Smith，他们为我们提供一个Linux兼容版的Grail+软件。我们还要感谢帕德伯恩大学（the University of Paderborn）的Marco Platzner和加州大学圣地亚哥分校（the University of California, San Diego）的Ali Irturk和Jason Oberg，他们给本书的初稿提供了宝贵的意见和建议。

本课题研究经费部分源于美国国家科学基金会，编号为：CNS-0524771和NSF Career Grant CCF-0448654的两项拨款，在此表示感谢。

Ted Huffmire
Cynthia Irvine
Thuy D. Nguyen
Timothy Levin
Ryan Kastner
Timothy Sherwood

作 者 简 介

泰德·霍夫曼（Ted Huffmire） 博士

海军研究生院 计算机科学系 美国 加利福尼亚州 蒙特雷坎宁安路（Cunningham） 1411 号 邮编：93943

电子邮箱：tdhuffmi@ nps. edu

美国加利福尼亚州海军研究生院计算机科学系的助理教授，他的研究领域跨计算机科学的两个专业方向，即计算机安全性和计算机体系结构。他的研究重点是面向硬件的安全性，以及专用芯片实施机制的研发。他在加利福尼亚大学圣芭芭拉分校，计算机科学系获得博士学位。他是美国 IEEE 和 ACM 的成员。

辛西亚尔·伊凡（Cynthia Irvine） 博士

海军研究生院 计算机科学系 美国 加利福尼亚州 蒙特雷 坎宁安路（Cunningham）1411 号 邮编：93943

美国加利福尼亚州蒙特雷海军研究生院信息系统安全学和研究中心（CISR）主任，计算机科学系教授。她的研究兴趣包括高保险度安全性。她在凯斯西储大学获得天文学科博士学位。她是 IEEE、ACM 和太平洋天文学会的成员。

秀·阮 （Thuy D. Nguyen）

海军研究生院 计算机科学系 美国 加利福尼亚州 蒙特雷 坎宁安路（Cunningham）1411 号 邮编：93943

美国加利福尼亚州蒙特雷海军研究生院计算机科学高级研究员，她的研究兴趣包括高保险平台、可信任的操作系统、动态安全性服务、多级安全性、安全性评价和安全性需求工程。她在加利福尼亚大学圣地亚哥分校获计算机科学学士学位。

提摩太·莱文（Timothy Levin）

海军研究生院 计算机科学系 美国 加利福尼亚州 蒙特雷 坎宁安路（Cunningham）1411 号 邮编：93943

美国加利福尼亚州蒙特雷海军研究生院副教授。他的研究兴趣包括高保险安全性体系结构和动态安全性策略的设计、分析和验证。他在加利福尼亚大学圣克鲁斯分校获得计算机科学学士。他是 IEEE 和 ACM 学会的成员。

瑞恩·卡思特纳（Ryan Kastner） 博士

加利福尼亚大学 圣地亚哥分校 计算机科学与工程系。美国 加利福尼亚州 拉乔拉 格立曼大道（Gilman Drive） 9500 号 邮编 92093

电子邮箱：kastner@ cs. ucsd. edu

加利福尼亚大学圣地亚哥分校计算机科学与工程系副教授。他的研究兴趣集中在嵌入式计算机系统，包括可重构计算体系结构，数字信号处理和安全性等许多方面。他在加利福尼亚大学洛杉矶分校计算机科学系获得博士学位。

提摩西·舍伍德（Timothy Sherwood）博士

加利福尼亚大学 圣芭芭拉分校 计算机科学系 美国 加利福尼亚州 圣芭芭拉 邮编：93106

加利福尼亚大学圣芭芭拉分校计算机科学副教授。他的研究兴趣包括计算机体系结构、专门研究可构造、监视和分析系统的新型高通量方法。他在加利福尼亚大学圣芭芭拉分校计算机科学和工程系获得哲学博士学位。他是 IEEE 和 ACM 学会的成员。

目　　录

第1章　概　　述

摘要：从蓝牙收发器到美国宇航局的火星探测器，现场可编程门阵列（FPGA）已经成为了嵌入式系统设计的主要方式之一。这种可重构器件，由于融合了硬件和软件的特性，因而能够在专用硬件的高性能和CPU的可编程性之间找到自己的应用空间，产生更好的效果。尽管这种灵活性可以让开发人员快速设计出原型机，并使设计性能接近于专用集成电路（ASIC）的嵌入式系统，但其可编程性可能会被敌方利用来中断关键的功能、窃听加密的通信内容，甚至摧毁芯片。构建高效灵活的系统，并确保其具有足够的安全性，是研究人员以及业内人士必须面对的艰难挑战。由于在可重构系统的设计过程中，通常要在设计末期才能发现其面临的安全问题，这就导致相关的系统只能通过系统深奥的难懂性[㊀]得到保护。本章将从安全性角度对现场可编程门阵列（FPGA）进行概述，重点说明这类器件在过去十年内为什么并如何成为现代计算机系统中最值得信赖和最重要的器件之一。本章还将讨论这类器件的角色转换（即如何从原型机平台转变成可应用的解决方案），介绍现代FPGA的架构，阐述由日益增加的用途而导致的安全衍生问题，以及可能适用于此领域的安全性方面的经验教训。

1.1　对FPGA日益增加的依赖

FPGA是很多领域关键设备的核心，在从无线接入点（WAP）到商用面部识别系统等几乎所有领域都有广泛的应用。与通用处理器的顺序执行方式不同，现代的现场可编程门阵列器件在每个循环中可以执行数百次乘法运算以及数千次加法运算，使其具有能够同时处理很多不同逻辑模块的计算能力。例如，采用FPGA的无线接入点（WAP）中，可能需要使用一个信号处理器、一个协议处理器，以及一个包调度器，所有这些可以集成在一个芯片中。另外，可重构硬件能同时在实验室及现场重新写入，因此可以保证较快的设计周期，相关的补丁甚至可以下载到已经开发完成的设备中（例如，可以根据需要，将错误修正或者功能增强补丁通过网络向手机或者无线接入点推广）。

由于集成了灵活性和计算能力，可重构器件已经推动了很多性能优异的嵌入式系统的发展[9,15,19,40,51,62]。很多可重构器件单位面积的速度和性能能够达到类似的

㊀ 这种难懂性源于非设计者对于设计的不了解。——译者注

微处理器的 100 倍[12,18,75]。卫星、机顶盒、入侵探测系统、电网、加密装置、飞机甚至火星探测器都需要通过现场可编程门阵列（FPGA）来实现相应的功能。据估计，仅仅在 2005 年一年内，就启动了超过 80000 项不同的商用 FPGA 设计项目[53]。这些器件在比特（bit）级上的可重构能力能够被用来实现所有高度优化的电路，从加密技术到快速傅里叶变换，甚至到完全自定义的多处理器系统。在本节中，我们将对其中几个领域进行介绍，并讨论 FPGA 器件在这些领域中的使用情况。

设计提示：FPGA 器件的优点。FPGA 非常适合用于嵌入式设计以及新型计算机体系结构开发过程中原型机的快速开发。随着专用集成电路（ASIC）生产成本的增加，FPGA 相对于通用型处理器的性能优势，以及 FPGA 和 ASIC 之间的较小的性能差异，都使得在实际系统中 FPGA 的应用日益增加。对于一些份额不大的市场（如高可靠系统），与专用集成电路相比，FPGA 同时具有成本优势和安全优势。例如，对于 FPGA 而言，敏感的设计绝对不会被送到可能造成设计失窃的晶圆代工厂中。另外，FPGA 器件提供的并行性使其能够用做由吞吐量推动的应用领域中的通用型 CPU 的替代产品。

1.1.1　航空航天用 FPGA

由于 FPGA 能够兼顾性能、成本以及灵活性等诸多方面，很多航空航天电子设备中都开始使用这种器件。例如，在联合打击战斗机[56]中、新的波音 787 梦幻客机[20]中、美国宇航局火星探测器[23,57]中都使用 FPGA 来实现重要功能。在这些应用领域中，FPGA 被用在驾驶员座舱显示装置、飞行管理装置、航空电子设备、武器制导设备以及飞行雷达设备中[2]。

设计提示：FPGA 基础。通过比特流来确定 FPGA 内部结构的配置位设置方式，FPGA 中的核是更大型嵌入系统的组成模块。反熔丝 FPGA 采用熔丝作为配置位，因此这种器件只能进行一次编程，之后数据不会丢失。SRAM（静态随机访问存储器）型 FPGA 采用 SRAM 存储单元作为配置位。Flash（闪存）型 FPGA 采用电可擦除可编程只读存储器（EEPROM）作为配置位。

电路可以是反熔丝电路、Flash 电路或者 SRAM 电路。其中基于反熔丝电路的 FPGA 为一次性写入器件，而 SRAM 型和 Flash 型的 FPGA 可以多次写入，可以在实验室写入或者在现场写入。Flash 型的 FPGA 还具有低功耗的优点。

设计提示：FPGA 选型。SRAM 型 FPGA、Flash 型 FPGA 以及反熔丝 FPGA 具有不同的安全性能[76]。反熔丝性 FPGA 的缺点是只能写入一次，优点是窃取设计时十分费力，必须采用破坏性的研磨及扫描破译技术（Sand-And-Scan Attack）。这包括拆除器件的封装、逐层研磨和剥离蚀刻的微电路，以及电子显微摄影，并通过这些来创建芯片的 3D 图像。由于反熔丝具有非易失性，比特流不需要由外部存储器载入，可以避免板级探测的破译，防止针对比特流加密机制的破译攻击。Flash 型 FPGA 同样可以将比特流存储在芯片中，这种设计方式也不需要从外部存储器载入比特流。因此，同样可以避免上述方式的攻击。尽管如此，和反熔丝 FPGA 相比，由于 Flash 存储器可以被修改，因此设计可能被改变。另外，与反熔丝型 FPGA 相比，Flash 型 FPGA 很容易被芯片探测技术破译，而芯片探测破译的成本比研磨及扫描破译的成本低得多。SRAM 型 FPGA 在上电之后，必须重新载入比特流，同时软存储错误[28]或者比特流解密机制实现过程中产生的缺陷会为破译者提供窃取设计信息的机会。在不断电的情况下，SRAM 型 FPGA 类似于非易失性的 FPGA 器件，此时不需要从非易失性的外部存储器中加载设计信息。

以军用航空电子设备为例，在这种设备中，单片机需要同时处理机密的目标信息，以及非机密的燃料和维护信息。航空电子设备涉及的其他多级安全（MLS）问题还包括传感器-发射器问题。与接到命令攻击相关目标的士兵相比，决定攻击目标的情报分析师具有更高级别的指挥决策权。在另一个多级安全（MLS）问题中，如果联军飞行员和其盟军组成编队一起飞行，则必须明确相关指令，确定哪些信息可以共享。在多级系统中，可以用 CPU 进行分配，使其处理某个特定秘密级别（或者某个秘密级别范围）的数据，同时为其设置一个安全标记（或者安全标记范围）。如果为每个秘密级别设置一个单独的器件，则会大大增加飞机的重量。为了降低飞机重量，可以考虑采用单一器件处理多个秘密级别数据。但是，如果对器件的安全性能没有经过仔细考虑，这种处理方式可能会非常危险。将不同秘密级别的信息分开需要经过仔细的设计。由于可重构系统常常缺乏存储保护、虚拟内存以及其他通用系统中经常采用的传统隔离方式，因此需要采取必要的安全措施来防止机密数据和非机密数据之间发生混淆。另外，同软件更新机制一样，在对器件进行远程更新时，保证安全十分重要，这能够有效阻止破坏行为。最后，由于所涉及的知识产权的敏感性，如何防止竞争对手或者敌人轻易地对芯片实施逆向工程也至关重要。

1.1.2　超级计算用 FPGA

尽管台式计算机的性能以不可思议地速度持续提高，但总是存在处于这些计算机能力范围之外的问题，在需要超强计算能力时，科学家和工程师最终不得不使用“大铁家伙”。很多超级计算机公司，包括 SRC 电脑公司[25,71]、Cray 公司[58]、以及 SGI 公司[67,68]都将可重构硬件集成到其系统中以便提高性能[10,16]。这种系统的一个最好例子就是 Cray 公司的 XD1 计算机系统结构，系统中每个机架由 6 片大型 XilinxFPGA（Virtex-4）以及 12 片 x86 处理器组成。加载到计算机中的应用程序包含了与其相关的可重配硬件的配置信息。尽管与微处理器相比，过去几代的 FPGA 在进行双精度浮点计算方面并没有成本优势[74]，但将其运用在定点运算占绝对多数的应用程序中，能够大大提高计算性能（大约 100 倍）。目前，FPGA 对于浮点运算提供了更多的综合支持。在超级计算环境中，所运行的代码或者数据通常具有敏感的知识产权，甚至是机密信息，因此需要相当安全的运算环境。此外，对于入侵者而言，超级计算中心是非常值得关注的目标。因此，超级计算中心还必须具有很强的物理安全等级。

SRC 公司的可重构计算机是采用 FPGA 对通用型处理器上运行的程序提供加速的一个例子[25,71]。此计算机通过传统的 Unix shell 界面登录。项目文件夹中可以同时包括 Verilog 代码和 C 语言代码（Fortran 代码）。一个编译文件可以针对 Verilog 代码激活 Verilog 代码编译器（产生一个比特流），针对 C 语言代码激活 C 语言编译器（针对 Fortran 代码激活 Fortran 编译器）。在 SRC 上执行程序时，需要将比特流载入到可重构硬件中，同时将可执行程序载入到通用硬件中。从安全性角度看，如果主机的操作系统、应用软件或者用户账号被盗用，则可能会将恶意的比特流加载到可重构硬件中。此恶意硬件可能会干扰应用程序的正常运行，甚至损坏硬件。由于 FPGA 是更大型系统中的一部分，安全分析必须考虑 CPU 和 FPGA 的相互作用。

1.1.3　用 FPGA 分析视频

FPGA 天生适用于复杂的高速信号处理应用程序，例如视频分析软件以及面部识别系统[59]。绝大多数这种类型的算法都是以大量的矩阵运算为主，属于吞吐量驱动算法，这就意味着对于这些应用程序而言，并行以及流水线运算能够大大提高其性能。本节以视频编辑问题为例来说明这些系统中由安全性衍生出来的问题。

修订过程涉及从资料（诸如文件、歌曲以及电影等）中删除敏感信息。视频修订可以用来从秘密文件中截取不需要保密的部分向公众公开，也可以用于保护个人隐私。视频修订的一个例子就是将监控摄像机拍摄到画面中人的面部进行模糊化处理。由于进行系统测试或维护的人员不一定具有查看相关人员面部的权限，进行

面部模糊化处理是非常必要的。IBM 公司开发出了一种叫做 PeopleVision 的视频隐私系统[65]。要在 FPGA 上实施这种系统，至少需要使用三个 IP 核：一个视频核用于视频处理，一个修订核用于将面部进行模糊化处理，以及一个以太网核用于将修订后的视频传输到安全保卫人员的终端。每个 IP 核都需要单独的外部存储器，同时必须对存储在外部存储器中的数据隐私进行保护。例如，视频核绝对不能绕开修订核将数据直接传输到以太网核处。由于应用程序经常采用由第三方开发的模块，因此构成嵌入式系统的模块的可信度是一个日益值得关注的问题，尤其是在软件和硬件开发过程中，知识产权的重新利用都是非常普遍的事情。

1.1.4 高吞吐量加密用 FPGA

在 FPGA 上实施加密具有很多的优点。分组密码需要很多的比特（bit）级运算，例如比特移位或者变换，这种运算能够在 FPGA 上有效实施。FPGA 还能够非常容易地更换算法参数，甚至将整套电路彻底更换。例如，如果数学家在某个密码中发现缺陷，可以很容易地用补丁版本更新 FPGA 的配置比特流。这些优点在 MD5[17]，SHA-2[70]，以及其他一些加密算法[17,37,43,55,60,64,66,70,77]中得到了应用。FPGA 对于公开密钥也非常有用，例如 RSA 加密算法，其中的基本运算是模乘运算，以及椭圆曲线加密算法，基本运算是点乘运算[27,29,48,54]。由于能够以较高的吞吐量对数据包流进行并行多规则搜索，可重构器件在网络入侵检测系统（IDS）中也得到了广泛的应用[4-6,13,14,21,26,36,72]。

绝大多数现代的对称密钥密码系统都具有针对一个给定的输入进行一系列重复的循环运算的特征。以循环运算为例，在循环运算中，将一个字中的 b 个比特位移动 n 个位置。在循环运算之前，此位原处于 i 位置，在运算之后处于 $i+n$ 模（mod）b 位置。使用软件实现这个操作需要多个指令，以便将比特位在字中进行移位，并将比特重新组合成字。与之不同的是，FPGA 可以通过重新排列连线的方式，使比特位到达其新位置处，进而简单地实现这些算法。几乎所有相关的工作都集中在采用 FPGA 达到较快的加密速度上，而很少关注可重构系统本身的安全性。由于加密器件通常需要处理秘密信息（例如密钥），因此对于攻击者而言，密钥具有很大的吸引力。保护措施不到位的系统非常容易受到各种定时攻击或者侧信道攻击，使得攻击者能够获得密钥信息。为防止攻击者为自己的利益修改加密算法，保证系统的完整性也至关重要。

1.1.5 入侵检测及防范用 FPGA

另一个广泛使用可重构器件且和安全相关的领域是网络入侵检测系统（IDS）。由于很多 IDS 系统需要对每个数据包中的每个字节都进行扫描，以确定其中是否存在已知的攻击方式或者可疑行为。入侵检测是一个对计算速度要求极高的难题。现

在的网络速度基本上已达到每秒数千兆比特（Gbit），而入侵探测系统必须随时能够跟上网络流量。这种最坏情况下的性能要求加上进行分析时采用的流水线方式，使得 FPGA 成为在该应用领域中几乎完美的器件。实际上，已经建立了很多采用 FPGA 的入侵检测系统[4-6,13,14,21,26,36,72]。这其中非常重要的一点是必须保证入侵检测系统（IDS）的完整性，并确保 IDS 系统不会被绕开（例如，以 IDS 核为中心来组织路由通信）。

1.2 FPGA 体系结构

FPGA 介于通用型处理器以及专用集成电路（ASIC）的某个中间位置。CPU 的典型特点是样样皆可，但样样不精。CPU 可以运行任何代码，为实现其通用性需要付出非常高的性能代价。编译器以及程序语言的出现改变了计算机科学，使得人类能够非常容易地编制计算机程序，这也使得我们经常忘记为获得这种通用性而付出的高额代价。另一方面专用集成电路能够通过在优化的电路中实现并行来获得非常高的吞吐量，但其加工费用却非常昂贵（费用每年都在增加），因此需要投入很多的资金。专用集成电路专用于某些领域，其所有的功能都是通过硬件实现的。与 ASIC 和 CPU 不同，可重构硬件能在不加工硅晶片的情况下实现自定义的电路。同时，与 CPU 相比，可重构硬件能够将吞吐量提高百倍[12,18,75]。本节将对典型的可重构芯片进行介绍，解释其性能优势，同时，本节还将介绍理解书中所述的 FPGA 安全机制所必须的背景知识。

1.2.1 可重构硬件的吸引力

可重构系统的起源可以追溯到（Gerald Estrin）在 20 世纪 60 年代于加州大学洛杉矶分校开展的相关工作。（Estrin）的“固定加可变结构计算机”[24]中包括一个由可重构硬件阵列增强的标准处理器。由于他的想法超越了他所处时代的技术水平，因此他只能将想法进行粗糙的近似处理。

在 20 世纪 80 年代中期，随着可编程逻辑器件的出现，人们开始再次关注可重构系统。当时这些可编程逻辑器件几乎毫无例外地被专用集成电路（ASIC）设计师们用作快速样机。这些器件使得设计师能够在可重构硬件上编译设计代码，以确定设计代码的功能是否正确。这种样机测试方法可以减少昂贵的 ASIC 投片次数，尤其在投片后发现器件功能不正确的情况下（有缺陷硬件）可以显著地节省开发成本。另外，随着样机测试技术的迅速发展，通过大规模仿真来验证设计功能正确性的需求减少了。如果设计代码编译后被加载到 FPGA 中运行，经测试证明其功能正确无误，则用该代码投片生产的 ASIC 器件的功能完全正确的概率极高。在这个时期，与 ASIC 相比，FPGA 的主要不足在于性能，例如延迟，功耗等。在 FPGA

中实现的设计在综合投片后成为一片功能固定的ASIC，并不具有FPGA动态重构能力。从固定功能角度来看，FPGA的性能永远无法超越ASIC[40]。尽管如此，随着ASIC设计成本的日益增加，以及设计规则变得日益复杂，很多用于提高ASIC设计系统性能的手段已经非常昂贵，或者非常容易出错，这缩小了ASIC和FPGA之间的性能差距[45]。

可重构系统的最大优势在于其灵活性。其灵活性不仅体现在允许根据输入参数、运行条件及升级情况进行在线重配置，同时还体现在允许单个器件的规模达到令人难以置信的容量。FPGA的大容量意味着晶圆工厂可以以非常低廉的价格（共用掩蔽层，大规模流水线生产等）、非常高效的方式（FPGA通常是技术领跑者，需采用最先进的光刻技术及工艺进行加工，同时还必须精心设计以减少变更和深亚微米效应）大量生产FPGA。

由于可重构系统具有定制输入数据的能力，所以用可重构系统实现的很多应用都能够以很高的速度运行。很多计算系统都可以在逻辑层面实现完全重构，同时很多器件都可以在体系结构层面上进行重构。另外，可重构核也被越来越广泛地用做嵌入系统的部件（例如，微处理器和可重构部件集成使用），同时还出现了集成有可重构部件的专用集成电路。由于上述原因，可重构计算机系统逐渐成为实施计算的一种重要组织结构[9,15,19,40,51,62]，它将CPU的通用性和硬件的空间计算特性集成在一起[19]。可重构系统采用可编程性以及通用架构来降低系统的复杂度、成本以及开发时间。

1.2.2 FPGA的内部结构

可重构器件有多种类型，FPGA是其中最常见的一种。通过综合工具和布局布线工具，FPGA可以在门级逻辑的基础上重新构造，这意味设计者可以把FPGA器件配置成为他想要的基于门级逻辑的任意电路。只要在设计过程中可以绘出数字电路示意图，即可开发出相应的FPGA来对其进行仿真。从物理的角度看，FPGA就是一组嵌入到灵活互连结构中的可编程门器件的集合，如图1-1所示。这些门器件采用查找表（LUT）作为计算单元，采用触发器作为时序单元，可编程内部连线作为路由单元，输入/输出模块（IOB）作为器件数据输入以及输出单元。由于任何逻辑门器件都有相应的真值表，因此可以通过下面的方式在FPGA中映射一个电路：通过合适的真值表配置查找表，同时通过传输晶体管设置开关盒内（switch-box）的配置位，确定哪根连线需要连接㊀。确定查找表以及开关盒应该如何进行编程的数据被称为配置比特流。FPGA的配置信息可以通过任意方式进行存储。例

㊀ 在晶体管中，输入信号不只施加到栅极上，同时还施加到漏极上。这种技术能够降低实现某种特定逻辑所需要的晶体管数量。

如，FPGA 可以使用可擦除可编程只读存储器（EPROM）/电可擦除可编程只读存储器（EEPROM）或者反熔丝（这是一种一次性写入技术，意味着在熔丝设置完毕之后，不能再对其进行重新设置）存储配置信息。有些结构使用了一次性写入技术，但大部分结构还是使用静态随机访问存储器（SRAM）作为可编程点。SRAM 使得 FPGA 具有易失性，这意味着在 FPGA 每次启动时都需要重新加载。最重要的一点是，SRAM 让 FPGA 具有可重构性，这是可重构计算的基础。SRAM 的配置位分布在整个 FPGA 内，和查找表以及互连开关盒（Switchable Interconnect）一起放置在 FPGA 的局部区域。内部配置文件较大时，重配置过程可能需要较长的时间，可通过配置缓存以及压缩来部分解决这个问题[33,50]。

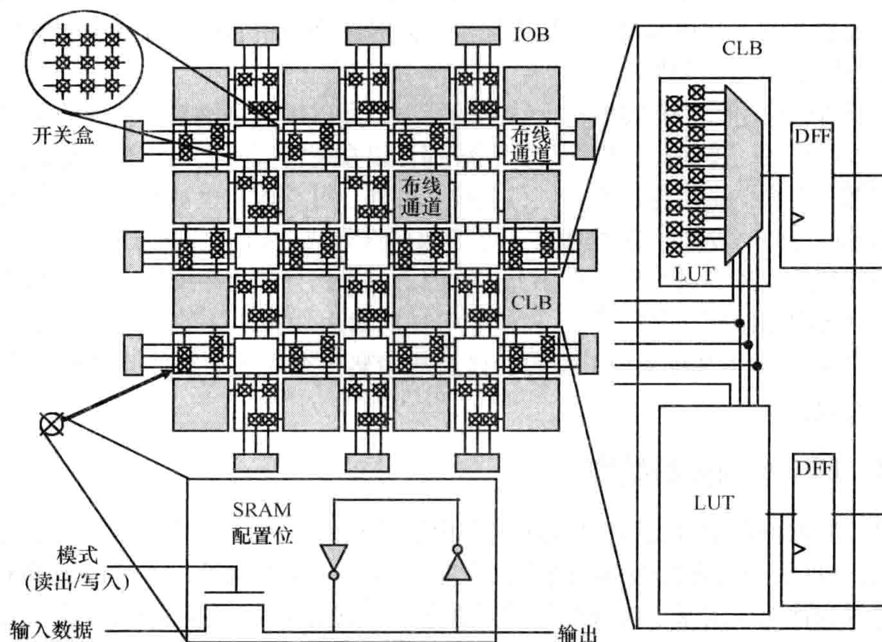


图 1-1　FPGA 的典型结构（多个配置逻辑块（CLB）被内部连线结构所围绕。每个 CLB 中都带有查找表（LUT），通过对其进行配置可以实现最基本的逻辑门功能。内部连线同样可以进行配置，通过其将 CLB 连接起来，进而能够将这些基本的逻辑门器件连接构成更加复杂的电路。FPGA 的配置比特流将明确 CLB 以及内部连线的配置方式）

静态 RAM（SRAM）每个单元（cell）中包括两个反相器以及一组晶体管，如图 1-1 所示。上电即可对单元进行数据的写入和读出。在不加电情况下，SRAM 单元中存储的数据将丢失。查找表采用 SRAM 单元作为编程位。由于能够实现任意的逻辑门功能，查找表具有通用性：N 输入查找表可以实现任何 N 个输入的电路

功能。需要采用 2^N 个数位来描述查找表，这能够实现 2^{2^N} 种功能。图 1-2 中提供了一个查找表的示例。SRAM 单元的尺寸限制了查找表的输入个数。基于大量的尺寸优化经验以及 FPGA 体系结构中其他方面的经验，查找表通常具有 4～5 个输入。现代 FPGA 体系结构通常被组织进入更大型的配置逻辑块（CLB）。一个 CLB 是由查找表、多路选择器以及触发器构成的复杂模块。另外，这些 CLB 中通常还带有自定义的逻辑器件，提高常见电路（例如，链式进位加法器中的进位链）的性能。尽管在最近几年内 CLB 的尺寸一直在缓慢增长，现在这些模块的尺寸依然比最小的微处理器还要小。

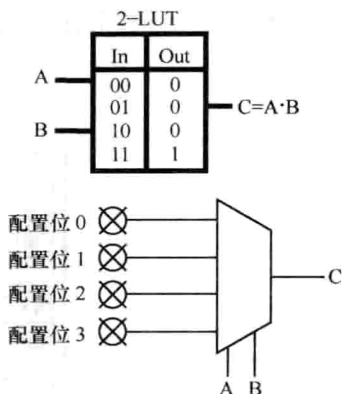


图 1-2 两输入查找表
（通过配置实现与门功能）

为了将小规模的计算模块连接在一起，FPGA 采用了岛状路由结构。通常岛状路由结构包括两个单独的部件。布线通道是一组向 CLB 提供可编程输入及输出连接点的晶体管。开关盒的作用为相邻的布线通道提供点对点连接，如图 1-3 和图 1-4 所示。在图 1-5 中描述了如何将内部连线分组到不同的布线通道中。布线通道中通常包括有长线，这些长线用于贯穿一行或者一列内的多个 CLB。通过长线能够实现 CLB 之间的快速全局连接，否则 CLB 必须通过多个速度极慢的开关盒进行连接，如图 1-6 所示。图 1-7 中描述了一个在每个开关盒连接点上带有六个传输晶体管的开关盒。HARP 开关盒是一种不同类型的开关盒，这种开关盒将硬线连接与可重新编程开关集成起来，以增加布通率[69]。由于电阻和电容以段长度的二次方形式增加，更长的路由段通常采用三态缓冲器，如图 1-8 所示。FPGA 的延迟及面积主要受布线结构的影响。在典型的 FPGA 中，约有 90% 的面积为内部连线（包括物理连线本身所需要的空间，以及将连线

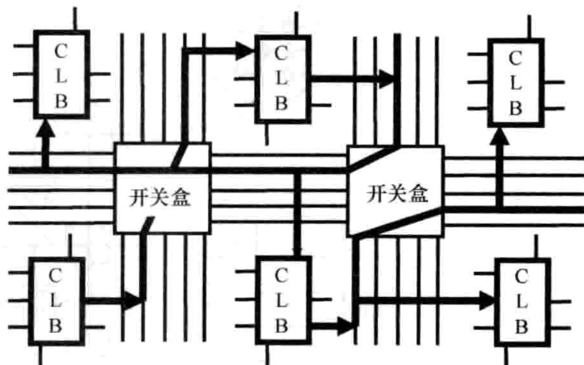
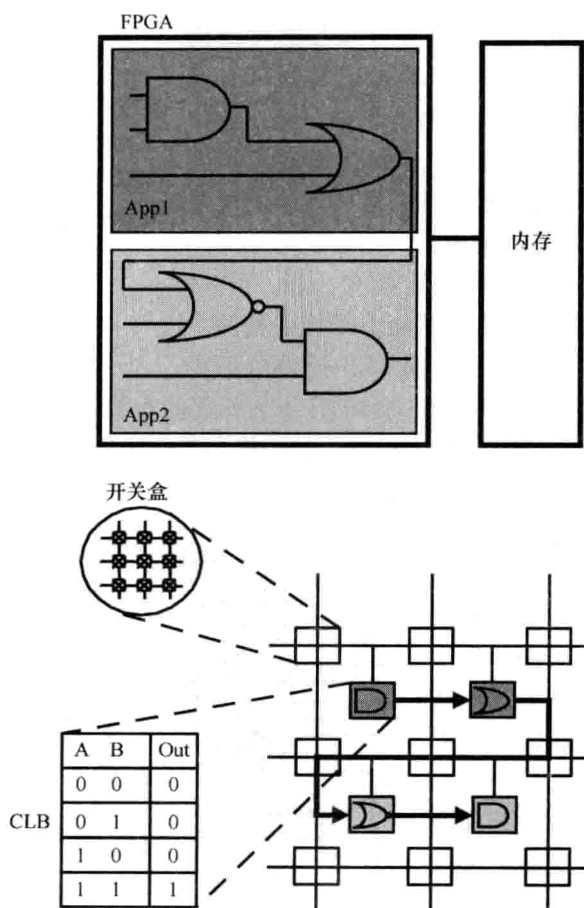


图 1-3 FPGA 内部连线体系结构（每个 CLB 模拟一个基本的逻辑门功能。通过将 CLB 连接在一起可以实现更加复杂的电路。内部连线负责将 CLB 连接起来。开关矩阵需要进行仔细设计，因为实现完整一个交叉开关时需要 N^2 个连接点。采用可重构硬件实现存储逻辑的效率较低，这主要由于能够取任何值的存储需要开关阵列为完整的交叉开关。由于这个原因，在 FPGA 中，在可重配置逻辑之间嵌入了硬线连接的存储单元（BRAM），甚至处理器）

连接在一起以便实现任意内部互联所必需的配置位占用的空间)。这种互联对于 FPGA 的配置非常重要，但这也会使得构建安全可重构架构变得非常复杂，这个问题将在本书后续章节中进行详细论述。



A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

图 1-4　在 FPGA 结构映射上绘制的最简单的电路图（App1 中包括一个与门以及一个或门，将其分别映射到一个 CLB 中。App2 中包括一个或非门以及一个与门，将其分别映射到一个 CLB 中。内部连线配置如下：App1 中与门的输出为 App1 中或门的输入，App2 中或非门的输出为 App2 中与门的输入；同时 App1 中或门的输出是 App2 中或非门的一个输入）

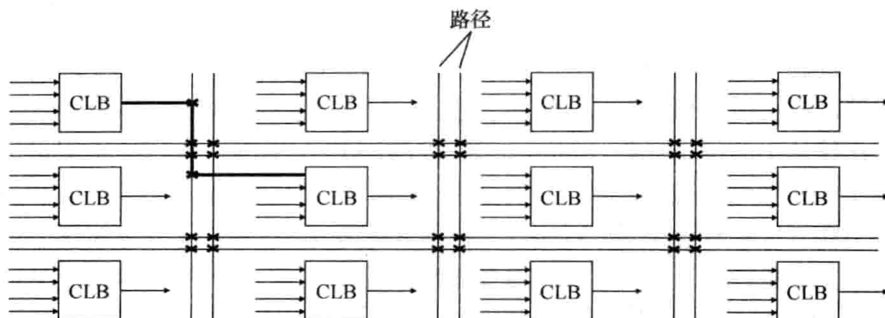


图 1-5 每个逻辑单元都输出一个数据位，不同单元之间的内部连线可以编程。互联路径被分为几组不同的通道

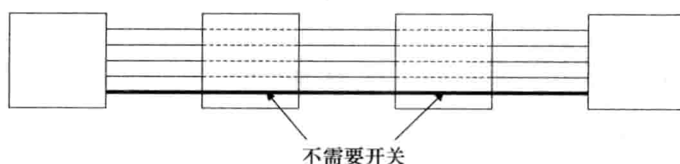


图 1-6 长线资源只需要在端部设置开关和连接，因此能够降低开关的数量，进而降低器件的总面积

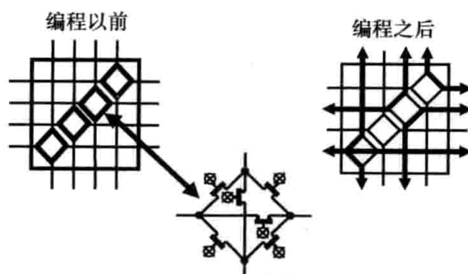


图 1-7 内部链接点上带有六个传输晶体管的开关盒（晶体管用作可编程开关，晶体管门由配置存储单元驱动。为了实现完整的交叉开关，需要 16 个晶体管）

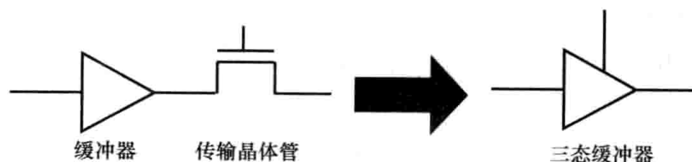


图 1-8 在 FPGA 内部结构设计过程中，较长的距离通常使用缓冲器，这主要是由于电阻和电容以距离长度的二次方增加

FPGA 采用比特流进行编程：在很多方面，比特流类似于传统微处理器系统中采用的二进制可执行代码。通过二进制数据对 FPGA 进行配置，使其实现具体的功能。比特流中包括所有 FPGA 的配置信息，包括开关盒的连接方式以及查找表中的数据。比特流中的每个比特（bit）位都对应着 FPGA 中的一个 SRAM 配置位。为了大概了解典型比特流的大小，我们采用 Xilinx XC2V6000 作为例子。这个器件大约需要 100 万位来对查找表进行编程，同时还需要 1500 万个配置位控制内部连线结构，以实现各种控制功能。这个器件的配置需要 1 ~3s 的时间，取决于具体的配置接口（例如 JTAG）以及器件支持的内部时钟周期，这些参数由开发人员进行配置。由于 FPGA 将这些比特（bit）位存储在易失性存储器中，因此在每次器件上电时，都必须重新加载这些比特流。正如本书后续的章节中提及的，这对于试图破坏基于 FPGA 的系统的攻击者而言，是非常有吸引力的目标。修改比特流就可以实现对电路的任意修改。另外，如果此比特流可以直接读取，不道德的人可以非常简单地将此比特流复制到另一套 FPGA 中，从而能够以极低的成本克隆相关的硬件。

1.2.3 设计流程

FPGA 系统的设计流程与传统的 ASIC 设计流程十分相似。通常采用硬件描述语言（例如 VHDL 或 Verilog）进行开发。然后，将由硬件描述语言表示的代码转换成许多布尔门之间的互联信号，再用一系列的输入信号对设计进行确认。然后，使用 CAD 工具翻译这些门，将其打包成更大的模块，直至可以映射到 CLB 中查找表的逻辑功能。接着，工具将确定在 FPGA 芯片上放置这些模块的最佳位置（这个步骤称为布局），并计算确定这些模块之间的连线通道（这个步骤称为布线）。然后对设计进行进一步分析，以确保其依然具有正确的功能，同时能够满足所有的时序要求。如果设计的功能或者时序存在问题，则应该对设计参数或者提供给设计工具的参数进行修改。由于这个步骤需要很长的时间，且保证大型设计的正确性是非常耗时的工作，因此大多数设计都十分依靠专用的（IP）核，这些 IP 核已经过充分的测试，被放在设计工具的组件库中，通过 IP 的互联可以满足具体的应用要求。

在软件工程中经常采用的代码复用技术同样也被用于可重构系统的设计过程中，以节省成本，并减少器件面世时间。由于需要较高的开发成本以及较长的开发时间，因此很少有人从头开始进行完整系统的设计。最常见的复用模块是软 CPU 核，例如 MicroBlaze。软处理器核其实只是配置 FPGA 器件，可在其内部实现通用处理器功能的比特流。将软处理器核与其他 IP 硬核或者软核（例如加密 AES 核和以太网核[⊖]）连接在一起可以组成一个可重构的系统。所使用 IP 核的来源有可能

⊖ 由于设计硬件模块的开发成本非常高，所有的公司都对其 IP 采取了严格的保密措施。在本书后续章节中，将讨论如何防止 FPGA 中 IP 核被窃取的几种机制。

导致重大的安全问题，即IP核是内部开发的还是从第三方获得的。所使用的IP核是通过诸如Base System Builder等开发工具产生的，还是从诸如opencores. org等开源网站上下载的？正如任意使用软件程序所面临的问题一样，虽然加密核的形式化验证是一项有用的技术[49]，但是无法通过形式化验证的计算来确定相关的硬件模块是否具有恶意，因为它是由专用的软件来实现的。例如，由恶意硬件设计师或者由恶意设计工具流程产生的IP核可能钻低级硬件机制的漏洞，窃取或者扰乱系统的逻辑功能。由于典型的设计中通常包括数以百万计的逻辑门器件，以及十倍以上的连接点，设计师需要一种方法以便能在器件上采用多种商业核来构建可信赖的系统，而不需要从头设计每个核（所有东西都从头设计同样并不能够保证成功）。大型复杂设计安全性的形式验证是一个严格且昂贵的过程，需要设计过程中使用过的所有核的蓝图（即HDL源代码）。由于商业秘密的存在，可能无法接触到相关的HDL源代码。另外，由于存在可扩展性问题，形式验证并不是包治百病的灵丹妙药。同时系统所采用的数学模型中可能也存在缺陷，甚至相关模型采用的数学证明过程中也存在缺陷。因此，要对于那些声明自身是安全的系统保持警惕。在通过数学“证明”安全的加密密码中也经常发现缺陷[8]。

设计提示：工具及核。在可重构系统的开发过程中，一个非常重要的方面是设计工具以及IP的来源问题，尤其是从公共网站上下载的设计工具以及核。一个非常好的做法是对开发工具以及代码库的安全性进行评估。后续章节中将对配置控制进行描述和讨论。

在典型的基于FPGA的嵌入式系统中，在同一片芯片上通常包括各种不同的核。可编程逻辑电路、硬核、硬的SRAM以及BRAM都布置在同一个FPGA之上，可以共享相同的外部存储器。在很多情况下，绝对不允许不同核之间通过共享资源（如外部存储、片上存储器以及总线）互相干扰或者互相窥视。这导致FPGA系统的安全设计面临着严重的挑战。

图1-9中明确了在相同的FPGA上可能出现的四种嵌入系统设计流程。

1）通过逻辑综合将硬件描述语言代码转化为网表，然后通过一个称为布局布线的流程将其转化为比特流。

2）通过系统级（ESL）电子设计工具（例如Celoxica），将用C语言（一种高级程序语言）编写的代码转化为处理器软核。

3）在另一种系统级（ESL）电子设计流程中，使用Xilinx公司的AccelDSP[34]将MATLAB[52]算法转化为硬件描述语言，然后将其转化为自定义的数字信号处理（DSP）核。

4）通过C语言编译器将用高级程序语言编写的代码转化为可执行程序，然后

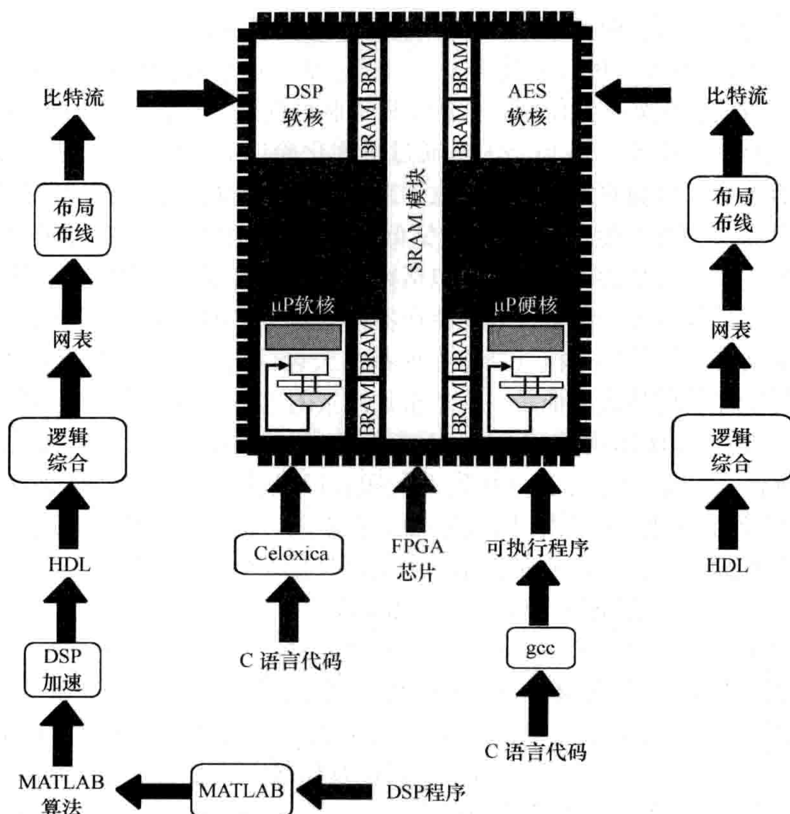


图 1-9　现代基于 FPGA 的嵌入系统（在相同的芯片上集成有多种具有不同来源的核。采用复杂的工具链来产生相关的核。AES 核可以通过将硬件描述语言代码转化为比特流的网表实现。DSP 核可以通过将 DSP 程序转化为 MATLAB 算法，再转化为硬件描述语言以及比特流的网表实现。图中描述的系统为电子系统级（ESL）设计示例。Celoxica 是 ESL 设计的另一种流程，在这种流程中将 C 语言代码转化为处理器软核。最后，采用诸如 gcc 等 C 语言编译器可以将 C 语言代码转化为可以在硬核上执行的可执行程序）

在硬核处理器上执行。

由于相关的工具是由不同的公司以及个人设计和开发的，这些工具也具有不同的安全等级。除了工具的来源之外，IP 软核（例如 AES 加密核）的来源同样是一个重要的具挑战性的安全问题。相关的核可以通以硬件描述语言（例如 Verilog）、网表㊀或者比特流的方式获得。这些核可能由开发人员设计，也可能是通过工具产

㊀ 逻辑门器件以及其内部连线的列表。

生的。

Xilinx ML507 评估平台，包含了一块评估板和一张附带的 DVD 工具盘，是使用不同工具进行系统开发的实例。ML507 中带有 Virter-5 FXT FPGA，在同一片芯片上集成了可编程逻辑电路以及 PowerPC 处理器硬核，同时 Xilinx 还提供集成综合（ISE）工具，用于通过硬件描述语言产生比特流。嵌入式开发套件（EDK）[78]用于创建定制处理器，允许设计师将其所需要的硬件模块或者其他外围设备连接到处理器上。EDK 同时还允许设计师对处理器进行配置（例如，改变频率、缓存大小等）。EDK 同时还提供了集成的基于 Eclipse 软件的开发工具包（SDK），用于对需要在 PowerPC 处理器硬核，甚至 MicroBlaze 处理器软核上运行的软件进行编译和调试。

由于上述各种不同的流程会最终产生可能包含有多个互相作用的核设计，因此根据系统的安全架构情况，最终获得的嵌入系统的可信度将由可信度最低的设计流程决定。例如，需要采用保护原语将加密核隔离，确保密钥不会被盗用。正如恶意的编译器会产生恶意代码，恶意的硬件设计工具也会产生恶意的硬件模块。被广泛使用的设计工具（例如，Xilinx、Altera 和 Actel）都没有能力确保其输出中不带有恶意功能。确定编译器或者硬件设计工具不含有恶意功能是一个公开的问题，至少和确定任意的计算机程序或者硬件模块中是否含有恶意代码或者恶意功能这个问题一样困难。

1.3　FPGA 安全问题的复杂性

在经济利益的驱使下，FPGA 在很多重要系统中的应用范围日益扩大，这使得设计师们不得不考虑安全问题。但在目前，从业人员依然没有可以遵守的设计规范。另外，由于嵌入式系统的资源限制，保证安全性更加困难[44]。除了将可编程器件应用于安全处理外，研究人员也已经开始考虑可重构系统本身的安全性。由于硬件设计能够直接从现场系统中复制，因此行业内部在如何保护知识产权[11,42,47]方面加大了投入，保证相关的 FPGA 器件只能由授权方进行更新[1,31,32]。尽管如此，只有很少研究人员会考虑 FPGA 器件中的恶意硬件模块[30]。Thomas Wollinger 以及 Saar Drimer 都针对 FPGA 设计安全性这个问题撰写过优秀的、全面的调查论文[22,76]。本书作者的调查论文在参考文献[35,39]上发表。在本书第四章中，将对比特流的加密以及验证进行更加详细的论述。

针对 FPGA 可能存在的多种攻击类型。在隐蔽信道攻击中，采用共享资源作为非法通信的手段。例如，恶意核可以对功耗进行调节，用来发送秘密信息给一直监控这些波动的另一个核[73]。部分 FPGA 支持比特流的远程更新，在这种情况下，最重要的一点是必须确保只有授权方才能进行这些更新。否则的话，攻击者可以上

传恶意逻辑指令，将 FPGA 配置在短路状态，进而改变系统的行为模式，或者损坏芯片[30]。尽管通过加密[11,41,42]、指纹[46]，以及水印[47]等方式能够防止知识产权失窃，但在对抗隐蔽信道攻击、侧信道攻击方面还需要采取更多的措施，并且进一步了解恶意硬件。

1.3.1 安全是一个难题

计算机安全一直是一个难题。尽管在计算机安全方面的支出逐年增加，每年的攻击次数也在增加。商用操作系统通常采用令人沮丧的出问题后补救的方法，进而陷入无休止的黑客攻击循环，供货商发布补丁之后，黑客又发现新的漏洞并实施新的攻击。在理想情况下，应该在一开始就保证系统的安全性。但是设计大型复杂并高度可信的系统是非常困难的。在这方面不存在任何灵丹妙药，因为每种安全技术都有其独特的优点以及缺陷。规则驱动的设计以及实现策略是采用多种互补的安全概念以及安全技术。目的在于增加攻击者的风险以及成本，使得攻击者在侵入每台计算机时都必须付出很大的努力。否则的话，一旦一点被攻破，系统可能就会像多米诺骨牌那样全面崩溃。依赖于一种防护技术非常类似于军事防御中的马其诺防线：攻击者能够从防护最弱的路径进入，然后完全绕开防线。若过度依赖一种单一防护方法来构建安全边界的系统，则在此安全边界被破坏之后将会出现严重的问题。系统设计师必须提防“所有安全边界范围内的资源都是安全的”这种假设。

加密技术是用于说明安全技术所具有的优点以及局限的基本例子。使用密码并不一定能保证系统的安全，但仍需要正确使用密码。所使用的密钥必须具有足够的长度，使其能够抵抗攻击者的暴力破解。加密处理器必须仔细设计，确保攻击者不能轻易地通过侧信道攻击确定密钥。必须对密码进行合理管理以及安全存储，加密文本和非加密文本绝对不允许混淆。加密算法尽管从数学的角度能够证明安全，但还是经常在其中发现缺陷。

设计提示：使用全面的安全原则。不能依赖于一种单一的安全策略。每种安全技术都有其独特的优点以及局限性。不要假设攻击者无法随意跨越安全边界。仅仅设置密码也不能保证系统的安全性，密钥需要得到合理的管理，还需要具有足够的熵，同时加密机制必须得以正确实施。历史上发生的加密和破解事件告诉我们，这个世界上不存在无法被攻破的密码。现在看来足够长的密码可能在未来就无法满足需求。不要假设由于数据已经经过加密处理在未来就永远无法被解密。最后，不要假设将数种较弱的安全技术结合在一起，就能为你的系统提供良好保护。

1.3.2 复杂度以及抽象

安全性问题之所以困难是因为系统的复杂度和规模。对于小型简单部件而言，形式化安全分析非常有效。但随着系统复杂度的增加，进行安全分析所需要投入的精力也会随之增加。复杂度随着分析目标规模的扩大而急剧上升。另外，独立状态下都处于安全状态的两个部件，将其集成起来之后可能就不再安全。摆脱这种复杂度的一种方法就是抽象。另一种管理复杂度的方法是对用户能够做的事情进行限制。例如，自动取款机（ATM）只提供很少几个按钮的有限界面，每种按键组合都能进行分析，以确定其不会导致不安全的状态。即使这样，在过去的时间内，ATM 的安全保护机制也经常被攻破[3]。

对系统安全性进行整体分析时，需要考虑 FPGA 嵌入到更大型系统中的情况。大型复杂的系统是由数个抽象层面构成的，芯片级别只包括单个 FPGA 器件中的部件，电路板级别包括焊接在电路板上的所有芯片。在更大型的系统（如个人计算机）中，可能包括通过主板连接的多块电路板，个人计算机本身又通过计算机网络（例如，以太网）组成多系统级别。整合问题指的就是独立运行时处于安全状态的两个系统，集成在一起后可能会变得不安全。对于必须同时采用可信赖部件以及商业货架产品（COTS）部件来构建的大型设计项目而言，情况会变得更加悲观。商业部件的重用是大型复杂系统构建过程中的一个重要方面。安全体系结构是一种在较高的抽象层次上对系统元素（包括计算部件以及安全部件）进行推理和分析的技术。在设计过程中，可以采用安全体系结构对部件进行组织，使系统强制执行安全策略，合理配置安全机制，进而确保所有的部件都遵循一个定义良好的接口标准。

为了解释芯片级、电路板级，以及网络级之间的区别，下面以 Xilinx ML507 评估平台为例进行说明。这个平台是一个 CPU 和可编程硬件可相互作用的开发板。开发板中带有一个 Virtex-V FXT FPGA 以及一个 PowerPC 处理器硬核，以及相应的逻辑电路。在 PowerPC 上运行的软件会和可编程硬件中实现的处理器软核相互作用，所有这些都在同一片芯片上完成。FPGA 芯片和电路板上的其他芯片以及其他部分相互作用，其中包括 DRAM、按键、发光二极管（LED）以及液晶显示屏（LCD）、USB、RS-232、以太网、DVI 等。在进行系统安全分析时，必须考虑在芯片级，电路板级，以及网络级上的所有相互作用（ML507 可以通过以太网和其他计算机进行通信，甚至作为一台网络服务器使用）。例如，在 ML507 上加载 Blue-Cat Linux 系统，可以在 PowerPC 硬核上运行，也可以在可编程硬件中的 MicroBlaze 处理器软核上运行。在这种情况下，操作系统、应用软件或者 MicroBlaze 处理器软核中存在的缺陷就可能被攻击者加以利用。

本书第 2 章中将讨论的引用监视器（Reference Monitor）概念，它是另一个非

常有用的抽象概念。引用监控器概念描述了一种小型的、能够自我保护的、持续运行的完美安全机制。这个概念是系统安全保障至关重要的一部分。这种机制将根据规定的安全策略来判断系统中的主体是否有权接触相关的资源。由于其规模较小，因此能够对其实施形式化完整的安全分析。这种机制无法被绕开，任何主体都不能绕开引用监视器。引用监视器有一个定义清晰的接口，所有主体都必须遵循。最后，这种机制具有防篡改能力，主体绝对不能干扰引用监视器的功能。这就是引用监视器具有自我保护能力的原因。

不幸的是，即使形式验证方法也不能包治百病。安全系统的形式验证首先需要有系统明确的数学模型，之后所有的证明都是围绕着此模型进行的。如果在模型本身或者证明过程中存在缺陷，则系统将不再安全。另外，在形式化描述和具体实现的转换过程中也存在问题。

设计提示：安全体系结构。为了管理设计的复杂度，应该为设计开发出相应的安全体系结构，以明确计算部件以及安全部件之间的组织方式。首先应该确定你所希望的安全体系结构需遵守的安全策略。对设计中与安全相关的部分采取保护措施。考虑 FPGA 嵌入到系统中的方式，以及和系统中其他部件相互作用的方式。不能仅仅因为已经采用了加密技术，或者诸如定理证明或者模型检查等形式方法就声明该系统是绝对安全的。

1.3.3 烘烤和修补的比较

构建能够抵御确定敌人恶意攻击的“高保障”系统是极其困难而且非常昂贵的。在系统整个生命周期中的每个阶段均应考虑安全问题（即“烘烤”思维），而不是出事后再补救（即“修补”思维）。系统的生命周期包括规范的确定、设计、实现、配置、操作以及更新等阶段。必须制订正确的安全策略。必须保证开发人员能够很容易地使用相关的安全技术，并保证终端用户能够容易地管理并运行相关的安全机制。必须对用户进行培训，使其能够正确地管理并运行系统。系统每个部件的设计必须正确无误。设计工具、硬件知识产权以及软件库不得被恶意篡改。

设计提示：在系统中建立。应该在产品的整个生命周期（包括设计、实现、配置、操作以及更新等阶段）中考虑安全问题。而不能在最后阶段通过补救的方式解决安全性问题。必须警惕现场系统已经通过了入侵测试，所以是绝对安全的想法。因为这种入侵测试通常是在产品开发周期的最后阶段进行的。

尽管存在严重的挑战，人们在数十年之前就已经知道如何构建具有高可信度的系统。Multics 系统（Unix 的前身）中就采用了数种设计方式来避免常见的缺陷，例如缓冲区溢出（这在今天依然是一个问题[38]）。这些设计方式中包括 PL/I 高级程序语言、硬件权限位、分段的虚拟地址、正向增长的堆栈，以及严格的开发方法。这个系统是针对大型计算机设计的，这些计算机早于目前运行在恶劣网络环境中的个人计算机。尽管 Multics 系统并不完美，但现今的操作系统供货商仍然从中吸取了很多的经验，直到现在依然没有可以广泛使用的完全值得信赖的操作系统。

即使在早期的分时计算机中，用户也对于将敏感性信息提交到功能安全性保证程度较低的计算机中这种行为感到担忧[63]。这种担忧帮助人们理解了这些机器到底在干什么，同时使得构建高保障系统的原则及技术要求变得越来越清晰。

1.3.4 FPGA 核的隔离

隔离是计算机安全领域中的一个基本概念，结合了隔离及受控共享这两个理念。加密系统是最早提出强隔离需求的系统之一，因为通过红线传输的明文信息必须和采用黑线传输的密文信息隔离开。Saltzer 和 Schroeder 将完全隔离定义为："将主体隔离在不同区间的保护系统，不同区间之间不可能进行任何信息交流或者相互控制"[61]。由于所有分区都完全隔离时，系统的功能比较有限，因此需要开发一种技术使不同部件之间数据的受控共享更加简单。

将 FPGA 模块映射到物理器件上时，其逻辑模块和内部连线之间可能发生严重的重叠。这就为攻击者通过创建一个模块拦截甚至损坏模块间的通信信息提供了可能，这就像利用网卡来拦截以太网中的数据一样。更坏的情况是，若短路配置被下载到器件中，则很可能导致器件的物理损坏。为了保证 FPGA 的安全性，必须将逻辑模块分隔在芯片上的不同空间位置上。在逻辑模块的分隔过程中，模块之间的内部连线（例如，总线）也需要进行仔细设计。

根据摩尔定律，带有 10 亿个晶体管的芯片足够容纳数百个精简指令集计算机（RISC）。计算机设计师无法在不违反上述基本限制的情况下提高单处理器效率，因此行业内部将希望寄托在多核结构之上。对于可编程系统而言，更大的整合度也是一个现实问题，部分 FPGA 能够容纳 8 个完整的 PowerPC 处理器软核。很多 FPGA 除了可以实现特定功能的可编程逻辑电路之外，还带有硬核。可重构片上系统（SoC）中同样带有硬件乘法器、SRAM 以及 BRAM 模块，这主要是由于特定的电路（尤其是存储器）在可编程硬件中实施时效率远远低于硬线连接电路。系统复杂度的增加使得保证这些嵌入系统的安全性变得更加困难。非常重要的一点是需要将这些部件进行隔离，以防止一个部件的弱点危及整个系统。由于这个原因，在系统具有较高集成度时，工程师需要在允许系统部件以受控的方式相互作用和通信的

同时，将这些部件进行隔离处理，以便提供保护措施。

为了保证隔离效果，对共享资源（例如外部存储器）进行管理是非常重要的。与通用系统中的 CPU 采用虚拟内存来保护存储器不同，FPGA 通常具有扁平存储层次，没有支持虚拟内存的操作系统。在缺乏上述机制的情况下，硬件模块可以读出并修改其他模块的存储器。另外，即使存在存储器保护原语，它们的设计也假设存在操作系统能够保证原语安全使用。在这种情况下，就必须采取一些措施来加强嵌入系统的安全策略。在本书第 5 章中提供了一种在可重构硬件中嵌入安全机制的方法，通过这种方法能够在硬件中强制执行存储器访问策略，通过此策略明确 FPGA 不同核允许共享的存储器空间，这种技术同时适用于外部存储器以及片上存储器。在本书第 6 章中，将提供一种利用 FPGA 的空间特性来强制执行安全策略的方法，相关的安全策略将明确不同核的隔离方式，包括核之间的隔离，以及核之间受控交互。对于专用集成电路而言，在其中加入创新性的安全机制是非常昂贵的过程（即行业从业人员通常不愿意将研究人员开发出来的安全机制引入到商用处理器设计中）。与 ASIC 不同，在 FPGA 的设计过程中，可以非常容易地引入安全措施。在 FPGA 中可以引入很多不同种类的安全机制。因此，从安全的角度来看，FPGA 所具有的可编程性可以视为优点，也可以视为缺陷。

1.4　本书结构

本书的目的在于为电子设计自动化（EDA）以及 FPGA 领域（包括企业、工业、政府研究实验室、研究人员以及从业人员）提供一种管理 FPGA 设计安全性的实用方法。书中将理论基础同实际的设计方法以及应对真实威胁的成功事例相结合。为了说明 FPGA 整个生命周期的不同阶段以及面临的威胁，本书从形式上处于最高级别的规范要求到最低级别的安全策略执行机制，对 FPGA 的安全问题进行全面分析。这部分内容包括计算机安全理论、程序语言、编译器以及硬件领域内取得的最新进展。最终提供一组多样化的、协同工作的静态和动态技术，以便能够通过商用部件获得具有鲁棒性、可靠性以及可信赖性的系统。

在本书第 2 章中对安全问题的基本原理进行讨论，说明了在过去数十年的计算机安全研究过程中，人们在高可靠软件领域获得的经验教训。第 3 章描述了硬件安全面临的挑战，其中包括恶意硬件、晶圆代工厂的可信度、物理攻击以及 FPGA 上不同类型隐蔽存储信道的探测以及预防。第 4 章讨论了与 FPGA 动态部分可重构相关的安全问题，以及采用部分可重构的设计中的安全管理技术。第 5 章讨论了用于防止外部存储器被用于在 FPGA 核间提供非法信息的存储器保护技术。第 6 章描述了一种用于防止 FPGA 不同核相互作用的空间隔离技术，以及一种用于防止通过总线以及直接连接点在 FPGA 不同核之间传输非法信息的方法。第 7 章提供了一个将

本书前面各章节中所述的安全保护技术运用到设计过程中的实例。最后，本书第 8 章对一些前瞻性问题进行了讨论。

参考文献

1. A. Abraham, It is I: an authentication system for a reconfigurable radio. M.S. thesis, Virginia Tech, August 2002
2. Actel Corporation, FPGAs for military, avionics, and high-reliability applications. *White Paper*, Actel Corporation, 2008
3. R. Anderson, Why cryptosystems fail, in *Proceedings of the 1st ACM Conference on Computer and Communications Security*, Fairfax, VA, November 1993, pp. 215–227
4. M. Attig, S. Dharmapurikar, J. Lockwood, Implementation results of bloom filters for string matching, in *Proceedings of the Field-Programmable Custom Computing Machines, 12th Annual IEEE Symposium on (FCCM'04)* (IEEE Comput. Soc., Los Alamitos, 2004), pp. 322–323
5. Z.K. Baker, V.K. Prasanna, A methodology for synthesis of efficient intrusion detection systems on FPGAs, in *Proceedings of the Field-Programmable Custom Computing Machines, 12th Annual IEEE Symposium on (FCCM'04)* (IEEE Comput. Soc., Los Alamitos, 2004), pp. 135–144
6. Z.K. Baker, V.K. Prasanna, Time and area efficient pattern matching on FPGAs, in *Proceeding of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays* (ACM, New York, 2004), pp. 223–232
7. V. Betz, J.S. Rose, A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs* (Kluwer Academic, Dordrecht, 1999)
8. A. Biryukov, O. Dunkelman, N. Keller, D. Khovratovich, A. Shamir, Key recovery attacks of practical complexity on AES variants with up to 10 rounds. IARC ePrint Report 2009/374, August 2009
9. K. Bondalapati, V.K. Prasanna, Reconfigurable computing systems. Proc. IEEE **90**(7), 1201–1217 (2002)
10. U. Bondhugula, A. Devulapalli, J. Fernando, P. Wyckoff, P. Sadayappan, Parallel FPGA-based all-pairs shortest-paths in a directed graph, in *Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium (IPDPS'06)*, April 2006
11. L. Bossuet, G. Gogniat, W. Burleson, Dynamically configurable security for SRAM FPGA bitstreams, in *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, Santa Fe, NM, April 2004
12. D.A. Buell, K.L. Pocek, Custom computing machines: an introduction. J. Supercomput. **9**(3), 219–29 (1995)
13. Y.H. Cho, S. Navab, W.H. Mangione-Smith, Specialized hardware for deep network packet filtering, in *12th International Conference on Field-Programmable Logic and Applications*, 2002
14. C.R. Clark, D.E. Schimmel, Efficient reconfigurable logic circuits for matching complex network intrusion detection patterns, in *Proceedings of FPL*, Lisbon, Portugal, 2003
15. K. Compton, S. Hauck, Reconfigurable computing: a survey of systems and software. ACM Comput. Surv. (CSUR) **34**(2), 171–210 (2002)
16. S. Craven, P. Athanas, Examining the viability of FPGA supercomputing. EURASIP J. Embed. Syst. **2007**(1) (2007)
17. J. Deepakumara, H.M. Heys, R. Venkatesan, FPGA implementation of MD5 hash algorithm, in *Canadian Conference on Electrical and Computer Engineering*, 2001
18. A. DeHon, Comparing computing machines, in *SPIE-Int. Soc. Opt. Eng. Proceedings of SPIE—the International Society for Optical Engineering*, vol. 3526, pp. 124–33, 1998
19. A. DeHon, J. Wawrzynek, Reconfigurable computing: what, why, and implications for design automation, in *Proceedings of the 36th ACM/IEEE Conference on Design Automation*, New Orleans, LA, June 1999

20. Design and Reuse Magazine, TTP controller IP in Altera's low-cost cyclone FPGA Families for Aerospace Applications, in *Design and Reuse Magazine*, 23 October 2007
21. S. Dharmapurikar, M. Attig, J. Lockwood, Deep packet inspection using parallel bloom filters. IEEE Micro **24**(1), 52–61 (2004)
22. S. Drimer, Volatile FPGA design security: a survey. Unpublished, Cambridge University, April 2008
23. Electronic Design Magazine, Actel FPGAs in Mars Rover, in *Electronic Design Magazine*, 6 August 2007
24. G. Estrin, Reconfigurable computer origins: the UCLA fixed-plus-variable ($F + V$) structure computer. IEEE Ann. Hist. Comput. **24**(4), 3–9 (2002)
25. O.D. Fidanci, D. Poznanovic, K. Gaj, T. El-Ghazawi, N. Alxeandridis, Performance and overhead in a hybrid reconfigurable computer, in *Proceedings of the 2003 International Parallel and Distributed Processing Symposium (IPDPS)*, Nice, France, April 2003
26. M. Gokhale, D. Dubois, A. Dubois, M. Boorman, S. Poole, V. Hogsett, Granidt: towards gigabit rate network intrusion detection technology, in *Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications* (Springer, Berlin, 2002), pp. 404–413
27. J. Goodman, A.P. Chandrakasan, An energy-efficient reconfigurable public-key cryptography processor. IEEE J. Solid-State Circuits **36**(11), 1808–1820 (2001)
28. S. Govindavajhala, A. Appel, Using memory errors to attack a virtual machine, in *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2003
29. C. Grabbe, M. Bednara, J. von zur Gathen, J. Shokrollahi, J. Teich, A high performance VLIW processor for finite field arithmetic, in *Proceedings of the International Parallel and Distributed Processing Symposium*, 2003
30. I. Hadzic, S. Udani, J. Smith, FPGA viruses, in *Proceedings of the Ninth International Workshop on Field-Programmable Logic and Applications (FPL'99)*, Glasgow, UK, August 1999
31. S. Harper, P. Athanas, A security policy based upon hardware encryption, in *Proceedings of the 37th Hawaii International Conference on System Sciences*, 2004
32. S. Harper, R. Fong, P. Athanas, A versatile framework for FPGA field updates: an application of partial self-reconfiguration, in *Proceedings of the 14th IEEE International Workshop on Rapid System Prototyping*, June 2003
33. S. Hauck, L. Zhiyuan, E. Schwabe, Configuration compression for the Xilinx XC6200 FPGA, in *Proceedings of Symposium on FPGAs for Custom Computing Machines*, 1998, pp. 138–46
34. T. Hill, AccelDSP synthesis tool floating-point to fixed-point conversion of MATLAB algorithms targeting FPGAs. *White Paper*, Xilinx Inc., San Jose, CA, April 2006
35. T. Huffmire, B. Brotherton, T. Sherwood, R. Kastner, T. Levin, T. Nguyen, C. Irvine, Managing security in FPGA-based embedded systems. IEEE Des. Test Comput. **25**(6), 590–598 (2008)
36. B.L. Hutchings, R. Franklin, D. Carver, Assisting network intrusion detection with reconfigurable hardware, in *Proceedings of the 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'02)* (IEEE Comput. Soc., Los Alamitos, 2002), p. 111
37. Y.K. Kang, D.W. Kim, T.W. Kwon, J.R. Choi, An efficient implementation of hash function processor for IPsec, in *Proceedings of the Third IEEE Asia-Pacific Conference on ASICs*, 2002
38. P.A. Karger, R.R. Schell, Multics security evaluation: vulnerability analysis. Tech. Rep. ESD-TR-74-193, vol. II, HQ Electronic Systems Division, Air Force Systems Command, Hanscom Field, Bedford, MA 01731, June 1974
39. R. Kastner, T. Huffmire, Threats and challenges in reconfigurable hardware security, in *International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, Las Vegas, NV, July 2008, pp. 334–345
40. R. Kastner, A. Kaplan, M. Sarrafzadeh, *Synthesis Techniques and Optimizations for Reconfigurable Systems* (Kluwer Academic, Dordrecht, 2004)
41. T. Kean, Secure configuration of field programmable gate arrays, in *Proceedings of the 11th International Conference on Field Programmable Logic and Applications (FPL'01)*, Belfast,

UK, August 2001

42. T. Kean, Cryptographic rights management of FPGA intellectual property cores, in *Tenth ACM International Symposium on Field-Programmable Gate Arrays (FPGA'02)*, Monterey, CA, February 2002
43. P. Kitsos, O. Koufopavlou, Efficient architecture and hardware implementation of the Whirlpool hash function. IEEE Trans. Consum. Electron. **50**(1), 208–313 (2004)
44. P. Kocher, R. Lee, G. McGraw, A. Raghunathan, S. Ravi, Security as a new dimension in embedded system design, in *Proceedings of the 41st Design Automation Conference (DAC'04)*, San Diego, CA, June 2004
45. I. Kuon, J. Rose, Measuring the Gap between FPGAs and ASICs, in *Proceedings of the International Symposium on FPGAs*, Monterey, CA, February 2006
46. J. Lach, W. Mangione-Smith, M. Potkonjak, FPGA fingerprinting techniques for protecting intellectual property, in *Proceedings of the 1999 IEEE Custom Integrated Circuits Conference*, San Diego, CA, May 1999
47. J. Lach, W. Mangione-Smith, M. Potkonjak, Robust FPGA intellectual property protection through multiple small watermarks, in *Proceedings of the 36th ACM/IEEE Conference on Design Automation (DAC'99)*, New Orleans, LA, June 1999
48. P.H.W. Leong, I.K.H. Leung, A microcoded elliptic curve processor using FPGA technology. IEEE Trans. VLSI Syst. **10**(5), 550–559 (2002)
49. J.R. Lewis, B. Martin, Cryptol: High assurance, retargetable crypto development and validation, in *IEEE Military Communications Conference (MILCOM)*, Boston, MA, October 2003
50. Z. Li, K. Compton, S. Hauck, Configuration caching management techniques for reconfigurable computing, in *Proceedings of Symposium on Field-Programmable Custom Computing Machines*, 2000, pp. 22–36
51. W.H. Mangione-Smith, B. Hutchings, D. Andrews, A. DeHon, C. Ebeling, R. Hartenstein, O. Mencer, J. Morris, K. Palem, V.K. Prasanna, H.A.E. Spaanenburg, Seeking solutions in configurable computing. Computer **30**(12), 38–43 (1997)
52. The Math Works Inc. MATLAB user's guide. *White Paper*, The Math Works Inc., Natick, MA, 2006
53. D. McGrath, Gartner Dataquest analyst gives ASIC, FPGA markets clean bill of health. *EE Times*, 13 June 2005
54. C. McIvor, M. McLoone, J.V. McCanny, Fast Montgomery modular multiplication and RSA cryptographic processor architectures, in *37th IEEE Asilomar Conference on Signals, Systems, and Computers*, 2003
55. M. McLoone, J.V. McCanny, A single-chip IPsec cryptographic processor, in *IEEE Workshop on Signal Processing Systems*, 2002
56. Military and Aerospace Electronics, F-35 Joint Strike Fighter uses Actel FPGAs for engine electronics, in *Military and Aerospace Electronics*, 1 September 2004
57. Military and Aerospace Electronics, FPGA processors keep Mars Rovers moving, in *Military and Aerospace Electronics*, 11 January 2005
58. K. Morris, Cray goes FPGA, in *FPGA and Structured ASIC Journal*, 5 April 2005
59. H. Ngo, R. Gottumukkal, V. Asari, A flexible and efficient hardware architecture for real-time face recognition based on Eigenface, in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, 2005
60. C. Paar, B. Chetwynd, T. Connor, S.Y. Deng, S. Marchang, An algorithm-agile cryptographic co-processor based on FPGAs, in *SPIE's Symposium on Voice, Video, and Data Communications*, 1999
61. J. Saltzer, M. Schroeder, The protection of information in computer systems. Commun. ACM **17**(7), 388–402 (1974)
62. P. Schaumont, I. Verbauwhede, K. Keutzer, M. Sarrafzadeh, A quick safari through the reconfiguration jungle, in *Proceedings of the Design Automation Conference*, 2001, pp. 172–177
63. R. Schell, Computer security: the Achilles heel of the electronic Air Force? Air Univ. Rev. **30**(2), 16–33 (1979)
64. G. Selimis, N. Sklavos, O. Koufopavlou, VLSI implementation of the keyed-hash message authentication code for the wireless application protocol, in *IEEE International Conference on Electronics, Circuits, and Systems*, 2003

65. A. Senior, S. Pankanti, A. Hampapur, L. Brown, Y.-L. Tian, A. Ekin, Blinkering surveillance: enabling video privacy through computer vision. Technical Report RC22886, IBM, 2003
66. Z. Shi, R.B. Lee, Bit permutation instructions for accelerating software cryptography, in *Proc. of the IEEE International Conference on Application-specific Systems, Architectures and Processors*, 2000
67. Silicon Graphics, Inc., Extraordinary acceleration of workflows with reconfigurable application-specific computing from SGI. *White Paper*, Silicon Graphics, Inc., 2004
68. Silicon Graphics Inc., SGI builds world's largest FPGA supercomputer, boosts nucleotide query performance by more than 900 times over 68-node cluster. *White Paper*, Silicon Graphics, Inc., 8 November 2007
69. S. Sivaswamy, G. Wang, C. Ababei, K. Bazargan, R. Kaster, E. Bozorgzadeh, HARP: Hard-wired routing pattern FPGAs, in *Proceedings of the International Symposium on FPGAs*, Monterey, CA, February 2005
70. N. Sklavos, O. Koufopavlou, On the hardware implementations of the SHA-2 (256, 384, 512) hash functions, in *Proceedings of IEEE International Symposium on Circuits and Systems*, 2003
71. M.C. Smith, J.S. Vetter, X. Liang, Accelerating scientific applications with the SRC-6 reconfigurable computer: methodologies and analysis, in *Proceedings of the 19th IEEE Parallel and Distributed Processing Symposium (IPDPS)*, Denver, CO, April 2005
72. I. Sourdis, D. Pnevmatikatos, Pre-decoded CAMs for efficient and high-speed NIDS pattern matching, in *Proceedings of the Field-Programmable Custom Computing Machines, 12th Annual IEEE Symposium on (FCCM'04)* (IEEE Comput. Soc., Los Alamitos, 2004), pp. 258–267
73. F. Standaert, L. Oldenzeel, D. Samyde, J. Quisquater, Power analysis of FPGAs: how practical is the attack? Field-Program. Logic Appl. **2778**(2003), 701–711 (2003)
74. K. Underwood, FPGAs vs. CPUs: trends in peak floating-point performance, in *Proceedings of the 2004 ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays*, Monterey, CA, February 2004
75. J.E. Vuillemin, P. Bertin, D. Roncin, M. Shand, H.H. Touati, P. Boucard, Programmable active memories: reconfigurable systems come of age. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **4**(1), 56–69 (1996)
76. T. Wollinger, J. Guajardo, C. Paar, Security on FPGAs: State-of-the-art implementations and attacks. ACM Trans. Embed. Comput. Syst. **3**(3), 534–574 (2004)
77. L. Wu, C. Weaver, T. Austin, Cryptomaniac: a fast flexible architecture for secure communication, in *International Symposium on Computer Architecture*, 2001
78. Xilinx Inc., Getting started with the Embedded Development Kit (EDK). *White Paper*, Xilinx Inc., San Jose, CA, 2006

第2章　高保障软件的经验与技术

摘要：本章介绍了一些从高保障系统开发过程中学到的经验，有助于读者理解在FPGA设计过程中必须建立的安全管理原则。这些原则包括风险评估、威胁模型、政策执行、生命周期管理、评估标准、配置管理以及开发环境等。

2.1　背景

从20世纪60年代早期开始，系统开发人员就一直关注未详细指明功能所导致的问题。这些问题包括在开发过程中引入的错误，以及由工程师添加的某些性能所引起的问题。通常这些添加的性能是善意的，不会造成任何问题，但在某些情况下，它们是恶意的，会造成严重的问题。

和软件相比，工程师们通常更相信硬件。他们经常假设硬件是值得信赖的，但事实却是，绝大多数的恶意软件都可以在硬件中实现。本章旨在介绍在系统实现过程中关于如何避免错误的一些经验。

2.2　恶意软件

恶意软件是指在功能上故意违反系统安全策略的软件。恶意软件的类型和可供恶意软件侵入的系统漏洞的种类非常之多。2007年在常见漏洞及披露（Comman Vulnerabilities and Exposures）项目提供的一份报告中，列出了41种不同类型的系统漏洞，其范围涵盖了从较弱的认证方式到跨网站脚本攻击[18]。本章主要讨论从高保障软件领域获得的经验教训，讨论仅限于两大类恶意软件：一种是在未授权域内执行的恶意软件；另一种是在已授权域内执行的恶意软件。

2.2.1　特洛伊木马

希腊人攻陷特洛伊的故事已经众所周知[37]。战斗持续了9年，希腊人仍旧无法攻破城墙并征服特洛伊。奥德修斯想出了一个计划，决定送给特洛伊人一份礼物并假装撤退。在将这份礼物（一只巨大的木马）拉到城内后，特洛伊人开始庆祝，此时希腊人从木马中突然现身，洗劫并烧毁了特洛伊城。不允许任何希腊人进入城内是一个安全策略，特洛伊木马是违反该安全策略的载体。在特洛伊人发现木马时，木马还在海滩上，是特洛伊人自己将木马拖到城内，并通过庆祝的方式将其

“激活”。

在计算机系统中，特洛伊木马是软件内隐藏的功能。隐藏有木马的软件能够提供一些其他用户需要的功能。如果用户下载或者以其他方式安装了未知来源的应用程序或功能，其中就可能隐藏有特洛伊木马。当特洛伊木马在用户应用程序框架内执行时，通常受到用户针对相关可执行程序权限的限制。攻击者对于恶意程序何时开始执行完全没有控制权。如果用户永远不执行带有特洛伊木马的程序，则木马也永远不会被执行。

尽管存在上述限制，在大多数系统中，特洛伊木马都能给系统的保密性、完整性以及可用性造成严重破坏。如图 2-1 所示的情形，Kathy 的一个文件中包含有 Sean 无权访问的信息。她对相关文件设置了访问控制，因此 Sean 将无法直接访问相关信息。不幸的是，Kathy 执行了一个带有由 Sean 其及同伙开发的含有特洛伊木马的应用程序。虽然合法的应用程序可以合法使用 Kathy 的文件，但木马同时也将她的信息写入到 Sean 的文件中。而 Kathy 却对此却一无所知。

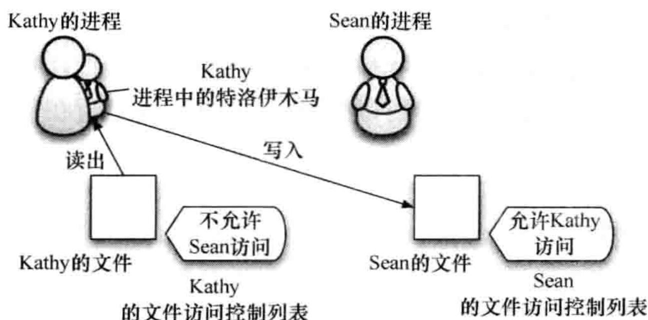


图 2-1 尽管 Sean 无权访问 Kathy 的文件，Kathy 进程中的特洛伊木马能够将她的所有信息写入到 Sean 的文件中

现在的特洛伊木马可能不会采取图 2-1 中所示的这种简单攻击行为。取而代之的是木马可能会将相关的信息发送到远程站点。这些应用程序的行为模式通常非常复杂，因此发送信息的机制可能会逃过内核级（kernel-level）审查机制的监控。

后文第 2. 5. 1. 2 节将对该问题展开讨论，实施强制性安全策略后，可以有效地限制特洛伊木马的活动。

2. 2. 2 后门

后门可以破坏软件的运行，最著名的例子是较早版本的 Excel 电子表格中隐藏着一个简单的飞行模拟器[100]。该模拟器由表格正常操作过程中几乎不会出现的条件组合激活。激活后，该模拟器会带领用户在一片阴郁的风景上空飞翔，缀有一个

数字不停滚动的墓碑，墓碑上刻有程序开发小组人员的名单。未指明的功能可以成为恶意代码的入口，由编译器在操作系统中插入后门[53, 101]就是一例。Anderson曾提供一个后门可破坏系统的范例[4]。尽管作者未曾试图使程序难以琢磨，但共计11行的代码却可以让攻击者能够在不被察觉的情况下访问整个Linux文件系统。由于攻击者可能不知道安装后门之后的理想攻击方式，更为谨慎的做法是开发一个级链的用于后续系统开发的后门入口，以便可以随时加载随意变化的恶意代码[58, 72, 81]。

后门和特洛伊木马之间存在下述区别。首先，后门能够由攻击者根据其意愿随时激活，而特洛伊木马需要受害者合作。其结果必然是，后门攻击者能够自由选择激活时间，通常通过一个触发器来激活或者使其不激活；对特洛伊木马而言，恶意程序的激活时间完全取决于受害者使用软件的时间。其次，一个低等级的后门可以绕过安全控制，而特洛伊木马的权限被它所依附的“受害者”权限所限定。最后，特洛伊木马通常在应用程序域执行，而后门的理想执行域为操作系统。

攻击者可以选择在系统生命周期的不同阶段（包括开发阶段和运行阶段）确定目标发动攻击。攻击者的目的是在系统中植入一个能够以无限特权执行的后门程序。Myers发现了在系统生命周期内适合后门攻击的若干时机[74]。系统的生命周期可划分为三个主要阶段：开发阶段、运行阶段以及退役阶段。系统开发阶段的攻击可导致无法正确地搭建系统，如高级别的策略及规范无法忠实地反映在系统实现的过程中。无意的错误以及有意植入的未指明的功能都可归入此类。

在系统运行阶段进行的攻击会使系统中的信息面临各种攻击风险，不精确的接口规范导致的毫无根据的假设都可以被攻击者利用。这些可被利用的缺陷可能是由不好的设计和实现所致。通过轮番攻击可以让攻击者获得重要的信息。若系统的构建方式允许对系统的运行状态进行操作，则可进行隐蔽信道攻击[60]。对硬件而言，攻击者可以利用侧信道攻击提取信息，这些信息包括功率以及时间分析等[56]。

在对上述阶段进行进一步细分之后，Myers确定在每个阶段都可能存在后门攻击，建议针对人员、过程以及工具采取严格的设计及开发方法学，并伴以生命周期保障作为一个整体方法，来减弱这些威胁。表2-1扩充了分析范围，将系统退役阶段包括在内。

表2-1 系统寿命期间后门攻击机会

阶段	威胁	阶段	威胁
设计	高水平设计师植入可利用的设计元素	安装	不受信任的安装程序在系统中植入后门，或者对系统进行错误配置
实现	在代码库中引入缺陷以及后门	运行	利用系统漏洞安装后门
发布	向产品添加附加物或者假的升级	退役	分析系统和媒体文件来提取信息

从 FPGA 后门发动的攻击可以发生在很多层面上，从开发阶段到运行阶段都可以对软件和硬件发动攻击。当然，设备中集成电路层面也会受到后门攻击。假如集成电路生产商不知道芯片用在哪里，则针对该芯片的攻击就有盲目性，没有把握。若该芯片有设计好的后门，则攻击就很有把握。卡尔格 Karger、谢尔 Schell[53] 和汤姆森 Tompson[101] 都断定：用于构建 FPGA 的工具可以为开发阶段的后门攻击提供载体。曲姆伯格 Trimberger[102] 描述了在对原始设计和实现后设计等效性进行无损确认过程中的一些挑战。他建议采用版图（Layout）与原理图（LVS）对比工具作为探测后门的手段。尽管系统的特性有限，但上述探测方式也极富挑战性。与 FPGA 相关的、被认为可能遭遇后门攻击的技术包括：设计工具的测试与确认、设计流程的验证以及 HDL 代码的静态分析。

2.3 保障度

系统功能上的安全机制指的是那些实现访问控制规则、登录机制、审查跟踪和各种安全管理员等功能的安全机制。而保障度是指系统的可信赖程度。仅某个用户可能信任一个系统并不意味着这个系统值得如此信赖。可信任系统意味着系统仅仅做了用户希望其做的事，没有做任何多余的事。若系统运行时出现错误，或者系统接口允许未授权擅用，则系统必有未指明的功能。此外，该系统还可能带有由恶意攻击者有意植入的功能。

系统的保障度要达到何种程度由其所有者决定。系统的信用或保障度可通过在整个寿命周期内谨慎管理的方式得以提升，系统的整个寿命周期包括从提出需求到系统退役的全过程。在任一个阶段，攻击者都可能试图进入系统，并在其中添加部分“特殊”功能。仔细的系统安全工程化、配置管理、值得信赖的交付机制、测试（包括内部和外部评审）以及形式验证等方法都对提升系统信用有帮助。

> 设计提示：保障度要求。安全性不是免费的。必须合理而明智地使用安全预算。在分析过程中，必须考虑到需要保护的资产价值及攻击者可以利用的资源情况。较高级别的信用需要在正确设计、实现、测试、交付、配置、操作及审查过程中付出更多的努力。有必要采取形式验证的方法来保证较高的保障度，但仅仅采用形式验证方法是不够的。

2.4 相称的保护

早期使用共享计算机的工作中，形成了很多关于计算机处理（即流程和计划）

及信息保护概念的模型，供货商采取不同的安全保护措施。计算机社团在零散的和有组织的活动中把计算机安全保护工作方面所发现的缺陷或错误[53]报告给供货商。今天的计算机供货商与用户间的关系在很大程度上依旧是这种关系，虽然很多供货商不再宣称其系统适用于安全领域[119]（唯恐他们必须为产品的安全性承担责任[27]）。供货商发售产品；用户发现产品中存在的漏洞并报告；供货商修复相关的漏洞并增加新功能；供货商再次发售新产品。补丁及新功能，尤其是两者相结合，可能会导致新的漏洞。尽管漏洞与补漏的方法并不令人满意，但这种方法能够避免系统安全性的前期投入。由于供货商发现用户能够忍受带漏洞的产品，因此这个循环一直持续。

将安全性植入产品内部的过程是昂贵的，因为此过程会增加设计、文档、配置管理及测试上的投入。尽管从长远看，这种谨慎的开发方法（参阅第2.3节相关内容）能够从整体上降低产品的维护成本，但这并不能说服急于将产品投入市场的供货商。在任何情况下，安全性都不是免费的，需要数据所有者更加关注的问题是安全性达到何种程度即可达到要求。

关于保护财产的普遍看法是，不应在保护措施上花费比需要保护的财产价值更多的钱。另一种准则是不要赌（即让其处于无保护状态）你输不起的东西。因此，可以根据需要保护的信息的价值，即在信息失窃（被盗用、被损坏或者使得所有者无法使用[61]）后给所有者造成的损失，采用成本效益以及风险分析方法来量化需要采取的保护措施级别。联邦紧急事务管理署（FEMA）[50]采用如下所述的通用公式计算财务风险，此公式假设威胁及漏洞等级数值在0~1范围内变化：

$$风险 = 资产评估价值 \times 威胁等级 \times 漏洞等级$$

对于信息保护而言，其他风险因素还包括在信息失窃的情况下资产价值损失的比例、相关信息对于潜在攻击者的价值（可能与其对所有者的价值不同），以及对漏洞的规避。对于为特定的资产及威胁提供的保护而言，可以将计算机系统的漏洞和无视漏洞的风险看作提供的负向保护。为了测定由IT系统提供的保护等级，目前已经提出很多种方法，其中包括各种不同的评估标准[14, 110]。

受到IT系统保护的资产中可能还包括人员，在对其进行估价时需要考虑诸如生命、自由等因素，这些因素与其货币价值之间可能存在主从关系。正如在本书下文中将详细讨论的，对资产的威胁进行量化也是非常困难的。

2.4.1 威胁模型

由联邦紧急事务管理署（FEMA）的风险计算公式可知，必须针对假设或确定的威胁设置足够的保护措施（即漏洞的消除措施），使资产面临的风险控制在可以接受的范围之内。如果没有威胁（例如，某人断言没有别人想要攻击），或者保护系统中不存在弱点（资产受到完整持续的保护），也就没有风险。然而，保守的假

设是攻击者的动机及资源，并与相关信息资源的价值成正比。换言之，价值越高的信息需要更可靠的保护。威胁模型是一种更系统的威胁评估方法[57, 66, 73]，包括下列各因素：

1）资产的性质-资产价值是否来源于其保密性、完整性及可用性；相关资产的价值是否会受时间影响（例如，部分知识产权或战略性军事数据可能需要保密几十年的时间）。

2）系统中可能存在漏洞的组件或者接口（攻击者可以通过这些漏洞访问相关的资产），在产品的整个使用寿命期间（包括设计、开发、交付以及维护等阶段），已知的针对相关的每种类型组件或者接口的攻击方式。

3）攻击者的动机，包括金钱、竞争优势、工业间谍、报复、名声以及声望名。攻击者的动机可能与资产的性质相关。

4）攻击者的能力（技术经验）、接触受保护系统的权限、资金以及其他资源（例如计算机使用时间及开发工具的可用性）。

除了主要的 IT 访问控制功能外，潜在的攻击面还包括程序以及自动生成的设计假设和支持策略的实例。支持策略包括：识别、认证、审查、物理安全以及用户的教育和审查。例如，性价比最高的系统攻击方法可能是社交及贿赂。

威胁模型不但能够用于组件以及产品的评估，也能用于系统部署环境的评估（这种情况下，威胁模型可能与威胁、资产及攻击者的特征更具有相关性）。下述内容仅限于针对组件及产品的威胁。

2.4.1.1 FPGA 接口

对于 FPGA 组件而言，威胁模型应该同时考虑内部接口以及外部接口。IP 核之间可以直接连接，也可以通过总线连接。网络接口可能有恶意的信息流。必须考虑 FPGA 可重构接口（例如，一些 FPGA 可以进行远程在线更新），同时还必须考虑共享计算资源的接口（例如系统及 Cache 存储器）。

2.4.1.2 FPGA 资产

FPGA 通常意义的信息资产包括：加密密钥、隐私信息以及专有逻辑设计。

2.4.1.3 针对 FPGA 的攻击

对于允许来源不明模块或者应用程序共享处理环境的系统而言，必须假设相关模块以及应用程序都是恶意的。在攻击分析中，必须包括产品安全评估及公开文献中描述的系统设计漏洞（如果有）的相关攻击。其他潜在的 FPGA 攻击包括：

1）通过 FPGA 的可重构接口上传恶意设计。恶意设计可以通过促使 FPGA 短路来破坏 FPGA 组件[41]。

2）利用使用过程中产生的影响或竞争共享资源。例如，如果 FPGA 的某个核使用计算资源的特性（例如，缓存区访问延迟时间、器件温度的变化情况或功耗变化情况）能够被另一个核检测到，则可能导致隐蔽信道或者侧信道攻击。

2.4.1.4　威胁模型中的其他要素

针对物理攻击（例如，读出或者摧毁加密密钥）可能需要在系统级、电路板级或者芯片级别采用篡改保护、探测和响应技术，具体取决于需要保护的资产的价值。同样，FPGA 制造设备及开发工具也能被用作攻击手段，设备和工具被敌方做了手脚，可使生产的 FPGA 设计弱不经风，这是第二层次的影响。本书第 3 章中，将对物理攻击方式进行更加详细的叙述。

2.5　安全策略的执行

为了系统的安全，必须针对该系统制订详尽的安全策略，再将此安全策略体现到系统实现过程的每个步骤中，以确保安全策略的落实。

2.5.1　安全策略类型

想要构建一个安全系统必须明确安全的含义。若由系统管理的资源都必须进行保护，则明确该安全系统属于哪种类型至关重要。安全性通常被理解为是一种策略。而信息安全通常是指保护信息并确保信息系统不被侵犯的一套措施。下面列出的是有关信息安全的五大要素，它们提供了实现信息安全总体目标的分类方法：

1）保密性（C：Confidentiality）——信息只对那些需要的人开放，防止任何未经授权的信息披露。

2）完整性（I：Integrity）——只允许具有适当权限的人员对信息进行修改，必须保护信息被非法篡改。

3）可用性（A：Availability）——有需要时，能以可靠稳定的形式使用信息。

4）认证性——信息的接收端完全了解相关信息真正的发送方或信息来源。

5）不可否认性——信息自发送端到接收端传输全过程的确凿证据能由接收者提供。

上述前三项与系统资源有关，后两项与通信有关，设计良好的协议及前三项不同的组合可支持后两项目标的实施。因此，我们将集中讨论前三个目标。Sterne[99]用非常通用的术语描述了一种组织安全策略。该安全策略转化到系统实现过程中后，将产生自动安全策略。自动策略通常是总体策略的一个子集，因为物理安全及人员安全已超出计算机系统可实现的范围。

我们知道，IT 系统的安全性应该在相关信息资源的保密性、完整性和可用性的安全策略框架内实现。对不同的系统，可以按不同的方式将保密性、完整性和可用性进行组合。例如，在实时控制系统中，高优先级事件的可执行性应处于最高地

位，而保密性应该处于从属地位。用于管理企业财务记录的系统可能更关注信息的完整性以及对相关责任进行控制。用于保护国家机密的系统应该首先保证保密性策略得以执行。其他的例子包括选举系统、健康记录系统以及员工薪酬系统等。安全工程师经常发现在 CIA 的这三个元素之间存在冲突，所有三种安全策略不能很好的被同时执行。其中可用性与保密性、完整性之间的冲突是最明显的。例如，在军用实时系统中必须同时对机密信息以及非机密信息进行管理。

在继续详细讨论策略实施之前，有必要介绍几个经过数十年已经证明行之有效的术语。

底层策略执行机制对资源进行控制，同时这些资源的子集可能会以抽象数据类型导出到机制接口处。其中可能既包括主动实体，也包括被动实体。这些可读写并且包含信息的资源称为客体。系统中的主动实体被称为主体，可以代表特定的用户或系统所有者。主动实体的典型例子是普通的应用程序进程，被动实体的典型例子是服务进程[59]。

系统安全性的先行者开发出了模型来描述策略执行机制的特征。例如，Graham 和 Denning[40]开发出了一种表状结构的模型，用于描述每个主体对客体的权限情况。如图 2-2 所示，矩阵中的每个单元内都明确了主体对客体的访问模式，允许相关的主体访问客体。

这些检查中所采用的参数来自于策略相关元数据，这些元数据与主体和客体具有相关性。元数据的性质由需要实现的策略类型决定。所有的策略均可分成下述两类：非强制策略和强制策略。因此，在一些执行任意访问控制策略的系统中，用户名或用户组与主体绑定，同时部分元数据（例如允许用户列表）将与客体关联。在执行强制访问控制策略的系统中，元数据可能由同时与主体和客体相关的敏感性标签组成。最为人们熟悉的标签就是在军事领域中，用于对信息密秘级别进行分类的标签，例如机密、密秘等。

如果一个访问控制系统带有允许修改控制策略的接口，将会导致一个有趣的后果，即无法开发用于判断专用保护系统中的信息是否会以非故意的方式泄漏的算法[42]。对于可以判断的保护模型，大量研究都探讨了其精确性[2, 86, 87]。

2.5.1.1 非强制策略

非强制策略是动态的，可以由无权限限制的主体在运行时加以修改，而强制性策略对于这些主体而言是不可变的。关于这两种策略的非技术类例子可以在刑事案件司法系统的判决指导原则中找到。对于很多罪行，主审法官能够对与具体案件相关的各种因素进行权衡，并决定与罪行匹配的惩罚方式。另外一种可选的替代策略是通过各种判决授权的条文规定，例如三振出局法案（事不过三，意在严惩累犯），来限制司法自由裁量权。由于存在这些限制，进行判决的法官对于惩罚方式没有选择权，即无自由裁量权。

客体 \ 主体	S1	S2	S3	S4	…	Sn
O1	R		W			
O2		RW	R			
O3		W	R	W		R
O4	R	R				R
O5	RW	RW	R	R		R
O6	R					
O7						R
O8	RW	R	R			
…						
	W					
Om	R	R	RW			

图 2-2 Graham-Denning 模型，在与主体与客体均相关的单元格内以矩阵的形式描述主体对客体的权限情况

执行非强制策略（即执行自主访问控制）的系统为应用程序或用户提供了可修改控制策略的接口。图 2-3 中解释了这种方式可能导致的问题。Kathy 的主体正在执行一个带有特洛伊木马的程序。由于执行了特洛伊木马代码，Kathy 的主体将利用运行时应用程序接口（API）来修改她的文件的访问控制列表（ACL），授予 Sean 访问权。此时，Sean 的主体即可读取她的信息，并将其储存起来以供未来使用。由于非强制策略是可以自主建立的，因此 Kathy 将无法保护她的信息不被 Sean 访问。

2.5.1.2 强制策略

强制策略除了不许随便更改策略外，还具有全局性和持续性。换而言之，在所有地方使用的策略都是相同的，不会因条件变化而改变。如果 Jim 的秘制烤肉酱配方在德克萨斯州是保秘的，则在柏林同样也应是保密的（即全局性）。而且，Jim 不能允许别人只在每周四上午 9 点到 10 点的时间范围内使用此烤肉酱的配方，因

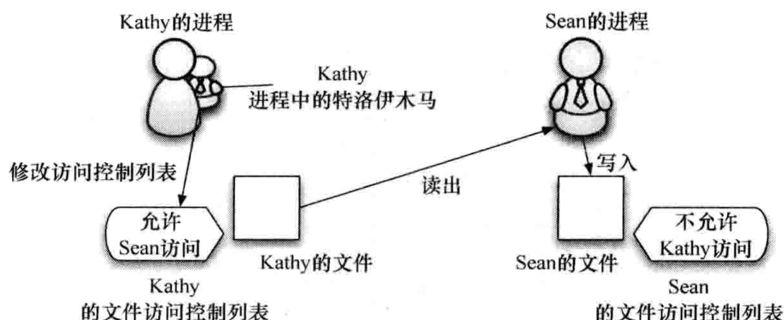


图 2-3　对 Kathy 的文件的访问控制列表进行了修改，使得 Sean 有权访问并保存她的信息

为如果别人在这段时间使用此配方，则不可能再对其进行保密（即持续性）。这些全局性和持续性的策略能够将信息以及有权访问该等级信息的人员划分成偏序（partial ordering）等价类的格阵形式[30]。最广为人知的强制性策略是军事领域采用的保密策略。在此领域内，通常根据某信息披露后给国家可能造成损失的大小来划分信息的密秘级别。军事信息的密秘级别通常分为绝密、秘密和非密。具有绝密权限的人员可以访问绝密、秘密及非密信息。具有秘密权限的人员有权访问秘密及非密信息。没有授权任何秘密级的人员只能访问非密信息。由于这些分类可以按照等级排序，因此这种策略被称为多级安全策略。然而，正如丹宁 Denning 指出的那样，只对密秘级别进行偏序（Partial Ordering）分类，有可能造成不可比较的等价类，因此信息组合也应纳入多级安全策略中，如图 2-4 所示。图中的箭头表示信息流的方向。因此，只有具有 {b，c} 或者 {a，b，c} 标签的读者才能够阅读标签为 b，c 的信息。

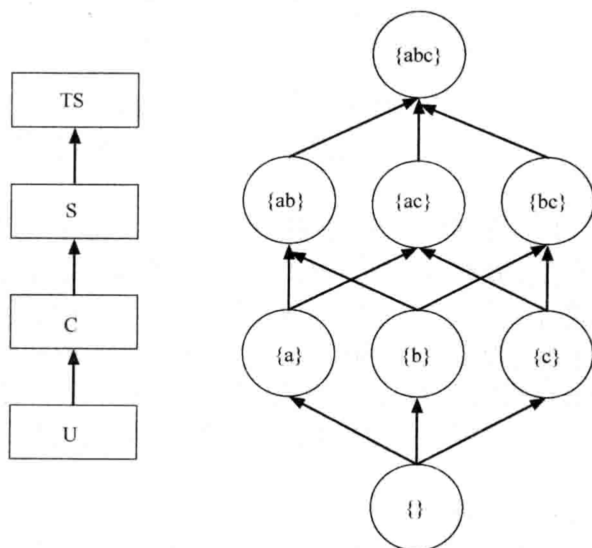


图 2-4　左侧方格内显示的是秘密级的顺序（U：非密，C：机密，S：秘密，TS：绝密），右侧显示的是秘密级层对应的集合

两个状态机模型分别体现了强制保密策略和完整策略的意图。Bell and LaPadula 模型[7,8]中描述了系统的安全状态及其三个属性：简单安全属性，*-属性以及自主性属性。本章只关注两种属性。如果某系统显示出简单安全属性，则

其实体只能够读取等于或低于其保密等级的信息。换而言之，它反映了典型的保密策略，即禁止密秘级别较低的人员阅读密秘级别较高的信息。系统必须保持的第二种特性是 *-属性，它很偶然地得到了这个不幸的名字。这种属性也被称为限制属性，用以反映信息限制的概念[60]，以及对付用户程序中的特洛伊木马。限制产生的结果是密秘级别较高的实体无法将信息写入密秘级别较低的信息库。

在强制策略的环境内，完整性要求信息具有双重保密性。完整性较高的信息只能由完整性权限较高的实体进行修改，然而任何实体，其完整性级别无论高或低(包括最低级别)，都应被允许读取那些完整性较高的信息。因此，在 Biba 模型[9]中包括了那些可以对观察、修改以及激活等操作加以限制的属性。

2.5.1.3 最小权限及其策略

最小权限原则是安全系统设计、实现及管理的基石之一。Saltzer 和 Schroeder[85]在一篇研讨会论文中对其进行了总结。在他们的论文中，提出了八条可以用于指导完整系统构建的原则。他们对最小权限的定义为："系统的每个用户及每个程序都只能在完成相关工作所必需的最小权限状态下运行。首先，此原则能够将由事故或者错误导致的损害控制在一定的范围之内。同时还能够降低操作过程中具有权限的程序之间相互作用的数量，以保证操作正确，确保无意的、多余的或不正确的权限使用情况尽可能地少发生"。通过硬件机制可以在系统内部创建保护模式，限制应用程序（相对于内核权限）的访问权限（参阅第 2.5.2.1 节相关内容)。

通过使用分层、模块化及信息隐藏等方式，可以在系统设计及实现过程中将最小权限原则反映出来。所有上述措施都是系统构建技术，在系统内部结构中使用上述措施，能够提高系统对渗透的抵抗能力。系统接口通过输出访问控制，粒度精细的执行域，这样主体就只能执行那些已授权的任务。

2.5.2 策略执行机制

为了执行访问控制策略，必须在系统中设置一个机制，来检查主体对客体的访问权限。

自从被提出之日起，基准监控概念[3]就成为系统执行安全策略的一个有用抽象模型。作为此类系统的理想化情况，这个概念可以作为完美的标准使用，保护机制的设计者也可以据此来测定其实施的保护机制的具体效果。

基准监控概念不涉及任何需要由系统执行的具体策略，也不针对任何具体的实现方式。这个概念清晰地归纳了理想的访问仲裁机制，并应具有以下三个特性：

1）访问仲裁机制应永远处于激活状态：每次访问都必须进行仲裁。如果不能满足此要求，就会存在某个实体绕开仲裁机制，从而违反安全策略的可能性。

2）访问仲裁机制必须具有防篡改能力。这样，渗透者就无法攻破理想的访问仲裁机制以关闭必要的访问检查。

3）访问仲裁机制本身必须足够小，能够进行分析和测试，并确保其完整性[3]。这就意味着此机制必须能够被理解，必须保证其执行预期的功能，不执行任何其他功能。

对访问仲裁机制进行的这些总结经受住了时间的考验，同时将继续作为描述安全系统设计实现抽象要求的有效工具。目前尚没有能够替代此概念的其他概念，此概念已经被证明有效，甚至在严格检查（close scrutiny）下也被证明有效。

Saltzer 和 Schroeder[85]描述了在较低权限的应用程序中保护最高权限系统元素所需的最低要求。其中包括权限位、存储器管理机制、特权功能的受控入口以及通过可靠方式将用户属性与代表用户执行的动态实体绑定。本书的后续章节将对上述各项要求进行更详细的描述。

2.5.2.1 优先指令、环和门

系统是按照权限组织起来的。硬件资源由权限最高的软件组件负责管理，这些软件组件在接口处组织并导出抽象数据类型，供权限低一级的组件使用。最简单的权限层级结构是提供两个权限域的双态处理器。其中的有权限域供操作系统或者内核使用，普通的应用程序使用无权限域。更加先进的硬件体系结构可以支持更多等级的权限域，例如 Intel x86 系列处理器能够支持四个硬件权限域[47]。

当企图访问资源、传送到不同优先域或者执行已选定的指令时，处理器检查任务内活动实体权限的能力是由硬件提供整体保护的机制中必不可少的要素。只有处于最高权限域内的实体才能够执行特许指令，如果某个应用程序试图执行特许指令，则硬件将会发出异常保护（标志），随后执行特许指令的请求将会被强行提交到异常处理程序中。对处理器状态可能造成影响的指令，如管理硬件存储器资源的指令、管理控制和其他寄存器指令、暂停处理器指令，以及执行有限数量的其他关键功能的指令，都属于特许指令。

出于性能的考虑，绝大多数的指令都可以被所有级别的权限直接访问。特许指令可以在经过虚拟化处理之后，将得到的抽象数据类型导出到内核的接口处。例如，内核能够以文件、段句柄或者其他对象的形式导出存储子系统的抽象数据。最后，部分特殊指令将被保留仅供内核程序使用，例如暂停指令。图 2-5 中描述了不同指令之间的区别。

2.5.2.2 存储保护、进程地址空间和虚拟存储

为保护自身以及彼此间的进程，系统内核必须对存储进行管理。存储管理指令的独占访问可以保证内核能够分配供其自身及由其创建的进程使用的存储资源。根据处理器类型的不同，此过程可能涉及对分区、分区表或者两者同时的管理。内核可以访问的地址空间包括整个处理器，此空间限制非内核应用程序的访问。在存储空间被访问时，硬件将对相关执行实体访问存储空间的权限级别进行检查。要保证存储空间访问过程有效，必须保证相关地址处于该进程的地址空间范围内，同时执

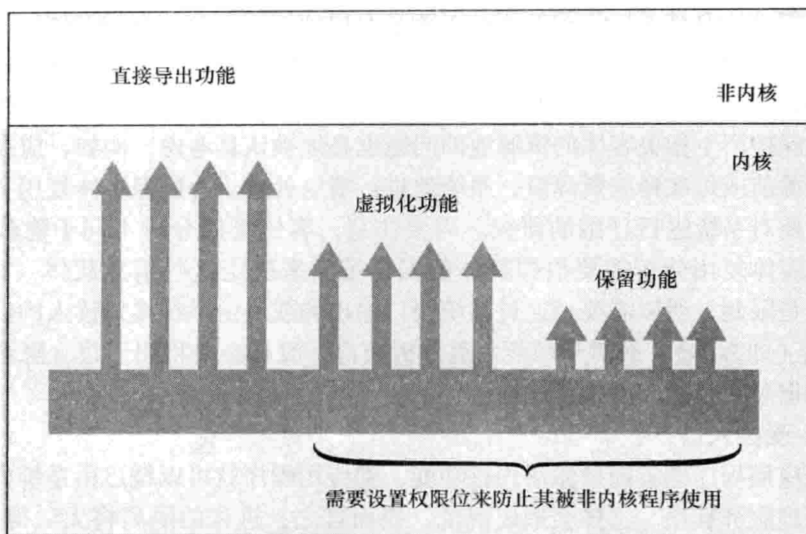


图 2-5　硬件指令可以直接导出到内核接口处，经过虚拟化处理后以新的抽象数据类型导出，或者保留仅供内核程序使用

行相关访问的实体应具有相应的权限级别。如果相关地址不在有效地址空间范围内，内核句柄应返回一个异常情况。

虚拟存储器又将地址管理问题的复杂度增加了一个新的级别。虚拟存储器能够让进程使用超过硬件可用存储资源的地址空间。每个进程的虚拟存储器都被划分为微小的、尺寸相等的页面。二级存储被称为交换空间，用于在进程的执行过程中维持相应的页面，并根据需要将相应的页面和主存储器进行交换。相关进程只允许访问分配给它的页面。为了保证性能最优，此虚拟地址到物理地址的映射采用硬件与软件相结合的支持方式。关于虚拟存储的详细叙述可以在很多的文章、资料和手册中找到[29, 43, 47, 95]。

2.5.2.3　客体复用机制

攻击者获取信息的一种方式是通过数据搜寻。在现实世界中，在垃圾桶中彻底搜寻可能含有敏感信息的纸片是最常见的获取有用信息的搜寻方式。对于数字化数据而言，数据搜寻同样具有吸引力。因此，不论执行了何种安全策略，都必须保证重新分配给不同进程的资源中，不含有与之前用途相关的信息。

客体是指系统中所有信息的载体，在安全系统实现过程中，描述此问题的常用术语是客体复用。由于客体是系统中无处不在的要素，同时客体经常被共享，因此已经开发出了很多保证客体复用的技术[76]。在被不同进程使用过程中，必须清空的存储器有主存储器、缓存、二级存储器、输入/输出器件使用的缓冲区以及各种

寄存器（包括所有被不同进程使用的可能带有以前进程残余信息的客体）。

确保客体的清空状态能被识别同时能够被有效管理需要采用系统的方法，如 Wichers 所述的方法[116]。相关信息客体是如何实现初始化、资源分配、资源释放以及如何保护产生相关客体的资源池的问题也必须被认真考虑。例如，应在每次重新分配资源前或每次释放资源后，系统地进行清空处理。为确保客体复用实现的完整性，需要对系统进行仔细的研究。需要注意，客体复用分析不同于隐蔽信道分析。因为客体复用分析需要根据具体的安全策略来决定这些信息载体（即客体）是共享还是限制，所以需要把通过系统接口输出的或经由系统接口读入的信息装入信息载体（即客体）。而对于隐蔽信道分析而言，没有必要将用于建立隐蔽信道的系统元数据装入可访问的信息载体。

2.5.2.4 受控入口

如果应用程序能够随意激活内核功能，则应用程序就可以绕过由系统内核提供的资源管理服务程序。这样会造成混乱，换而言之，进程的隔离将无法得到保证，系统就不能保护内核资源不受应用程序的调用。在这样情况下，对安全策略的任何期望都将化为泡影。为了给内核提供一个保护层，以确保内核功能只有在得到允许的情况下，而且只有在想要实现的内核功能时才能被非内核实体调用，系统内核必须具有一个可以对所有访问请求进行控制的机制。

为了完成这个任务，必须有硬件支持。最简单情况下，需要在应用层激活一个特殊指令，将控制权转移到内核区域的一个特殊位置。在此位置上，内核将对参数列表中的预定位置进行检查。除常规的参数外，还将提供一个请求功能标识符。然后，系统内核将对相关参数进行验证，确保指针和地址属于相关应用程序的范围，而不是内核地址。最后，系统内核将控制权转交给处理相关进程的功能模块。

上述机制对于只有两个权限域的系统而言是足够的，如果系统具有多个权限域，则会产生新的问题。如果所有的内核确实都已被低级权限域的指令激活，则无法确定中级权限域是否处于被保护状态，不会被低级权限域有意或者无意误用。针对此问题，较好的解决方法是采用门机制来控制向内的调用[77]，即调用门[47]。这些门机制位于每个权限域的边界上，对级别较低的权限域发往级别较高的权限域的执行指令进行控制。可以对这些门进行设置，可以设置成发送给级别较高的权限域的指令必须通过一系列串联的门进行，也可以设置成可以跳过中间级别的权限域。这样能够让每个权限域将其功能导出到系统选定的级别较低的权限域中。例如，内核可以只将特定的内核管理功能发送到下一级权限域中，而不进入到级别更低的权限域中，这也让系统构建人员能够从外部向内核提供值得信赖的代码以便进行资源管理。相反的，保障度较低的应用程序将不能激活内核的管理功能。中间级别的权限域可以将抽象数据类型导出，同时需要通过相关的门机制来激活处于权限域接口上的类型管理器。

2.5.2.5 用户属性绑定：可信路径

由于与代表用户的客体属性绑定是访问控制决策的基础，所以清晰明确的用户识别和认证策略对于安全系统至关重要。出于这方面的考虑，Saltzer 和 Schroeder 将可信赖的识别及认证技术纳入到系统保护的基本要求中[85]。

假设用户、用户识别及认证机制都是值得信赖的，用户如何才能确定输入的重要安全信息不在人机接口与 I&A（识别和认证）机制期间被中间人或者其他恶意实体截获呢？这需要在受到保护的用户通信信息和值得信赖的系统之间建立一种无法伪造的联系。这种联系被称为可信路径。用户通过安全认证密钥激活可信路径。换而言之，此密钥可以是一个单一的密钥，也可以是不同密钥的组合，或其他专门用于建立可信路径的输入方式。安全认证密钥信号被系统中的可信机制接收，以可信的方式向用户显示 I&A 接口。后续的人机相互作用可以受到保护，同时用户也可确定密码以及其他重要的认证信息已经受到保护。

值得注意的是，可信路径不能对密码的输入造成限制。也就是说，其他的重要信息可能也需要采取类似的保护措施。银行账户资料及信用卡号、大型金融交易的在线确认、某些电子健康记录的访问入口或者某些高价值交易等都是通过可信路径进行激活的。

在单一平台上，可信路径要求用户接口只能通过可信的机制向用户展示。这就意味着使用来源未知或者可疑的大型图形用户接口库将无法归入可信路径机制范围。作为系统总体安全架构的一部分，可信路径至少应和系统中用于执行安全策略的部件具有同等保障度。

可信路径通常指人与计算机间的接口。当然，在分布式系统中，系统间通信的保障度也是非常重要的。可信信道是用来描述系统间通信保护措施的术语。在分布式系统中，可信路径以及可信信道可能都需要进行加密处理。这个过程中必须注意，确保使用的加密功能以及密钥管理机制都具有较高保障度。

系统用户认证可根据如下所述属性进行：相关用户的物理特征、相关用户已知信息、相关用户所拥有的物品。根据用户物理特征进行系统认证时，通常采用用户的生物学特征。其中包括许多不同认证方式，常见的例子包括指纹识别、声音识别、视网膜扫描，虹膜扫描以及面部识别。使用生物特征作为访问控制方式需要首先录入用户的基准特征。然后将提出登录要求的用户的生物特征与之前录入的基准特征相比较。由于在生物特征采集过程中存在很多变数，因此两次采集的生物特征很少完全符合。需要采用统计学的方法来确定接受或不接受相关登录请求。在采用生物特征作为用户识别方式的过程中，面临很多重要的挑战，例如安全性问题，可扩展性问题，隐私问题，互操作性问题，以及社会方面的问题[35, 49]，这些问题都必须加以考虑，确保相关用户的生物特征能够在保密的情况下使用。根据用户已知信息进行认证时，密码是用户已知信息，可以用作访问系统的依据。密码不存在上

述采集过程中出现的波动和变化问题，但容易被猜到和强制破解攻击。同时在密码的复杂度和用户的可接受度之间必须做良好的平衡。通过用户所拥有物品进行认证的实例包括用户向系统提交身份证进行认证。由于用户所拥有的物品容易失窃或者丢失，因此这种认证形式通常还配套有附属的第二套认证机制。为了保证只进行有效的认证，同时降低认证过程的复杂度，相关组织机构通常将不同的认证方式组合起来使用。采用两种不同方法来进行用户认证的方法被称为双因素认证。在进一步扩展认证方式数量的情况下，很容易将这种认证方式进一步扩展为多因素认证。

由于存在很多可供使用的认证机制，系统的开发人员必须从物理和技术的角度考虑相关认证机制的有效性、便捷性、使用环境、成本及其可维护性。

如果用户持有一个可以登录所有系统的密码，则盗用此密码会导致所有系统中受到保护的信息都处于危险状态。为了消除这种威胁，建议用户针对不同的账号设置不同的密码。但随着账户数量的增加，用户必须记住大量密码。密码系统可能出现的第二个问题是在一次特定的任务过程的不同阶段，用户必须进行多次认证才能获得相关服务。在这种情况下，可以采用单点登录机制来解决用户的烦恼。尽管单点登录具有独特的优点（如能够减少密码数量，容易记住和输入），但也有着严重的缺陷（如在账号被盗用的情况下会造成非常严重的损失）。

2.5.2.6　非强制（即自主访问）控制执行机制

非强制访问控制通常通过两种类型的机制执行，即访问控制列表和权能。

访问控制列表（ACL）以明细的方式列出相关主体针对不同客体（例如，文件、文件夹或者设备）具有的权限。在访问控制列表中的每个条目中，都包含有代表相关实体（例如，个人用户或者用户组）的名称以及相关实体具有的权限。由于访问控制列表需要面对大量类似的用户（例如，所有心理学系 101 班新入学学生），因此将用户进行分组能够简化工作量，同时降低管理员犯错误的概率。范围最大的用户组是所有人，在很多系统中称其为公共用户。最简单的访问控制列表出现在 Unix[5] 及其后续系统（例如 Linux）中，在其中使用了很短一组权限位来确定访问相关文件的权限，即所有者、用户组以及公共用户。经过几十年的发展，很多商用系统已具有十分复杂的访问控制列表，系统权限管理者可对表中的每个用户的权限做明确的定义。

访问控制列表不仅可以用于获得访问权限，还可用于拒绝特定主体的访问。就像考试后向学生发布试题的标准答案一样。如果 Andy 不在城内，未能如期参加考试，而他必须在本周内补考，则在他补试前，教授将拒绝 Andy 查阅试题标准答案的请求，考完后才允许他查阅标准答案。因此访问控制列表中就有授予班级所有学生构成的用户组读取权限的条目，同时还有一条附加条目，通过此条目可以拒绝接受 Andy 的访问请求。聪明的读者可能已经发现，Andy 是有权访问答案文件的班级用户组中的一名成员，因此必须确定关于访问控制列表中相关权限优先级的规则。

在这种情况下，如果针对个体用户的访问控制列表权限的优先级高于针对用户组的权限，则教授就可以拒绝 Andy 查阅答案的请求。Lunt 详细讨论了在确定访问控制列表权限优先级过程中需要注意的问题[68]。

访问控制列表中包括的最简单权限可能仅为读出、写入及执行权限。因此不同用户以及用户组对于相关客体可能拥有一种或者多种上述权限。由于非强制访问控制通常应用于专业领域，因此可能需要创建其他类型的访问许可。例如，在有些情况下，授予特定用户对相关文件（例如，日志文件）进行添加的权限可能比较有用。在这种情况下，将只允许用户在文件的末尾处执行写入操作。为了实现添加功能，基本保护系统将使用读出与写入组合的权限管理方法。在没有用户信息情况下，系统将打开日志文件，将写入指针设置在文件结尾处，然后在文件中写入下一条日志记录。也可能系统或者应用程序的接口明确不向用户授予读写权限。

如果访问控制列表中包括针对相关客体的许可，那么针对访问控制列表的权限是如何管理的呢？这个问题非常重要，因为允许对访问控制列表进行修改的运行接口是将其识别为非强制访问仲裁机制的元素。在有些情况下，可能授予相关用户针对访问控制列表的控制访问权。具有控制相关客体访问权限的用户或用户组可以分配、授予或者拒绝其他的访问权。这些控制访问权可能划分得较为粗略，例如，向不同的个体用户授予针对访问控制列表的不同访问权（如读出权限）。另外，控制这个概念可以进一步向上延伸几个级别，因此特定的用户可能享有控制访问权的控制权。这种访问权的丰富组合形式让相关的组织机构能够更加灵活地设置非强制访问控制来满足具体要求。

新客体被创建后，非常重要的一点是保证每份访问控制列表的初值都能够满足预设的安全策略要求。在部分系统中，可能需要设置一份模板文件，以便针对每个新创建的客体提供默认的访问控制列表。上述默认列表适用于整个系统，也可以更细地划分，例如在客体是文件的情况下，可以针对不同的文件夹设定不同的模板。Lunt[68]对默认的自主访问控制权限进行了分析，从无访问权（最低权限）到完整的访问权。在最小权限框架内，无访问权或者授予有限的访问权可能是更为明智的选择。

访问控制列表极具吸引力，主要因为它能将所有与具体客体相关的权限都集中在一起。在这种情况下可以很方便地对安全策略进行管理。尽管如此，值得注意的一点是，对安全策略进行的修改不会立即生效。访问控制列表只在相关客体打开后，进行实际的读写操作前进行一次访问权限检查。访问权限检查的结果将被存入缓存中，主体保持客体打开时，首次访问获得的权限保持不变。因此，除非采用一些更加专业的机制，撤销访问权不会立即生效，只有在用户尝试再次访问相关客体时才会生效。

2.5.2.7　权能系统

权能是实施非强制访问控制的另一种方式。权能系统这个概念首先由 Dennis

以和 Van Horn 提出[32]。在权能系统中，对客体的访问权限清单与主体相关。除了定义控制权限之外，权能还提供一种对客体进行命名的方式，能够为基于权能的寻址提供基础[36]。例如，用户登录到系统中后，将会产生一个对代表用户的主体具有约束力的初始权能组。随着执行过程的进行，主体可能获得其它额外的权能。在主体试图访问客体时，参考验证机制（RVM）将会对权能中含有的访问权限进行检查，如果要求的访问包括在权限范围内，则将允许其继续访问。因此，在主体拥有针对某个具体客体的权能时，可以根据权能中包括的访问权限对客体进行访问，所有主体都需要做的事情就是明确其权能。Levy[65]对权能系统进行了详细讨论。权能机制最著名的一个实现示例是在性能防护（CAP）系统中所采用的操作系统。

由于权能系统在主体之间分配其针对每个客体的访问权限，同时相关的权限将存储在每个主体的初始化数据中，因此撤销相关的访问权限将会出现问题。另外，如果主体能够复制并存储权能，则权能撤销问题又会进一步恶化。并且不存在可以进行检测的中间位置来确定哪些主体能对某个客体进行潜在访问。相反，每个主体的权能列表都必须进行检查。如果某人决定取消主体对某个客体的访问权限，则可能需要对系统中所有的权能列表进行检查，确保权能已经完全撤销。另外，和访问控制列表相同，如果主体正处于访问相关客体的过程中，则权能的撤销不会立即生效。Redell[80]提出了一种解决此权能撤销问题的方法。在需要执行强制性策略的系统中，权能会导致更麻烦的问题，因为在典型的强制性策略系统中，不会区分访问权限和授予访问权限的权限[10]。权能系统可以进行扩展，使其支持由 Karger 和 Herbert 提出的基于格子的安全策略[51, 52]。

尽管权能系统能够实现，但这种系统的复杂度也是人所共知的，另外这种方法还缺乏概念简单的策略执行机制，这些都是使得这种方法在高保障度方法领域内前景黯淡的原因。尽管如此，权能系统还是一直被人们广泛关注，例如参考文献[92, 118]。

2.5.2.8 强制性策略执行机制

将不同的域隔离需要对子系统进行隔离，同时还需要设置控制不同域之间资源共享的机制。

2.5.2.9 强制机制类型

安全内核将内部敏感性标签和导出资源绑定在一起，同时根据内部策略模块中所定义的标签的偏序对主体访问其他资源的权限进行控制[88]。标签空间可以支持保密性和完整性策略，就像无等级分类方式一样[69]。一个安全内核通常提供硬件支持的环状抽象[91, 93]，同能够支配受信任的客体[89]。此环可以分隔一个权限域内的进程。因此，一个主体就是一个进程-环对。到目前为止，所有高保障度安全内核都使用段存储器，这提供了持续的基于硬件的进程-本地存储器保护属性[33, 38, 89, 90]，而不是基于存储器分页机制的动态全局的硬件属性。

安全内核能够通过专用于给定敏感性级别的网络设备，或者通过多级设备对外部通信进行管理和控制，这些多级设备将针对每个网络协议数据实体（例如，数据包）绑定一个敏感性标签。安全内核通常能够在运行时支持完整资源和资源可配置。

分离内核[83]，有时也称为分区内核[67]，将其导出资源的集合映射到不同的分区：

资源映射图：资源——→划分

多种主体资源以及客体资源可能会被划分在某给定的分区中，但分区只是一个抽象概念，本身并不是主体，相对于分区间的流动而言，给定分区内的资源同等对待。一个分区中的主体可以访问其他分区中的资源。分离内核强制实施分区的隔离，同时允许这些分区中的每一个主体在映射到分区空间时产生流，这些流组成了分区之间的流动（可以在不同的分区间或者在同一个分区间）。被允许的分区间流动可以建模为分区流矩阵，矩阵中的条目将说明分区流的模式，和前文所述的图2-2类似。

分区流：分区×分区——→模式

模式说明了流动的方向，因此，为

分区流（P_1，P_2）=W

意味着 P_1 中的主体有权在 P_2 中写入任何信息。资源在分区之间的配置方式以及访问控制规则或者分区流规则将以配置数据的形式传输到分离内核中，在系统初始化过程中，分离内核将对其进行解释。由于配置数据的正确性对于预期安全策略的执行至关重要，因此通常需要采用配置工具来构建分区流规则。尽管这工具不属于内核本身，但它能够帮助安全管理员或者系统集成者组织复杂的数据，并进行视觉化处理。这能帮助用户，确保输入反映了预期的安全策略。

最小特权分离内核（LPSK）与基本分区内核比较，有两项重要的优点。首先，根据论文《Separation Kornel Protection Profile（SKPP）》[46]的描述，LPSK 增大了赋予主体特权的粒度。其次，与分区内核不同，LPSK 扩展了监视器引用的功能，使监视器成为所有内部分区流的控制点。另外，除了分区内核中的资源图（resource_map）以及分区流（partition_flow）功能之外，LPSK 还能够以主体资源流矩阵（subject-resource flow metrix）的方式支持最小特权原则。

主体资源流：主体×资源流——→模式

因此，用主体资源流矩阵规则来取代分区流矩阵的相关规则是可行的[46]。尽管如此，一个更严格的解释是，只有在两个矩阵都允许的情况下，LPSK 才被允许产生特定的资源流。这种解释更为直观，也更有可能在系统的执行中实现正确的配置[63]。

允许流（主体，资源，模式）

——→模式∈主体资源流（主体，资源）& 模式∈分区流（主体分区，资源分区）

SKPP 要求如下：

1）每个安全配置中都有不同分区之间流的基础偏序标识，以确保严格地执行 MLS 策略。

2）被允许在分区间产生除上述基础流之外的其他流的主体都可被视为可信主体。

图 2-6 中说明了如何通过以上两项策略的实施，将 MLS 安全策略的粒度进一步细化。基线偏序情况如图 2-6a 所示，同时图 2-6a 中还明确了分区的偏序情况：信息可以沿着图中粗线箭头的方向流动，从 P1 到 P2，以及从 P2 到 P3，此时处于分区中的主体为可以产生信息流的实体。最小特权如图 2-6b 所示。在这种情况下，只有特定的主体能够产生信息流，如图中细线箭头所示，信息流指向或者离开特定的资源。例如，S_{22}只能从 O_{21}以及 O_{22}处读入。可信主体也许是只对一些特定信息进行降级的专用工具，如图 2-6c 所示，在这种情况下，允许产生从 S_{32}到 O_{12}的信息流。在所有主体均为可信主体的情况下，可以保证系统的安全策略意图得到满足，图中的阴影部分以及虚线箭头是指基阵情况下信息流动方向相反的方向。根据策略规定，只有在 S_{32}到 O_{12}的信息流被允许的情况下，明确的分区规则才会允许从 P3 到 P1 的信息流。

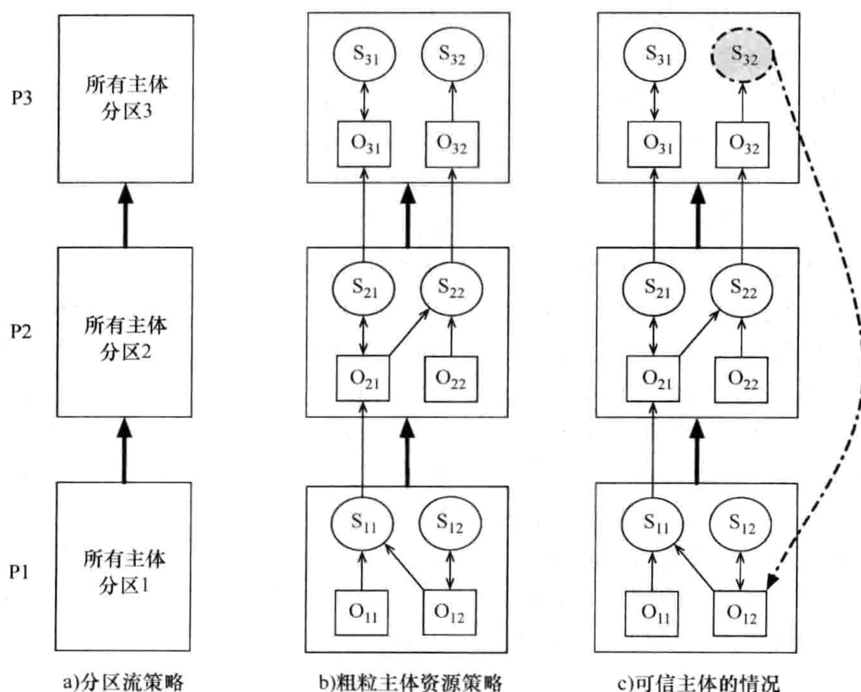


图 2-6　SKPP 策略

关于上述三种方法各自的优点综述，可以在 Levin 等人的论文[64]中找到。

2.5.2.10 审查机制

通过审查机制可以提供安全相关事件的记录。如果其中包括动态规则检查，则在出现安全违规事件时，审查机制会发出警告。首先必须制定相应的安全策略，确定需要审查的内容。例如，审查内容可以包括：只对某个特定客体的访问进行审查；对系统中的所有活动都进行审查；对某个特定敏感性级别的主体活动进行审查；对选定系统调用的使用情况进行审查等。由于安全管理员以及其他可信任人员会参与到对安全而言非常重要的活动中，因此应持续对其行为及活动进行审查。另外，还需要有很好的精简工具，否则产生的大量审查记录将毫无用处。

入侵检测系统（IDS）属于动态形式的审查。Anderson 曾在 1980 年针对入侵检测系统的下一步工作提出过建议[19]。他的论文于 1987 年发表，其中提供了一种通用的 IDS 模型[31]。从那时起，人们开发出很多用于网络入侵检测的系统（例如 Snort[96] 和 Bro[79]），还开发了很多以主机为基础的入侵探测系统（例如，Tripwire 系统[55]）。在 Snort 和 Bro 系统中，对网络流量进行密切监控，以发现表征网络入侵前奏或者初始阶段的行为。在后来的系统中，主要检测单一平台上的行为，以便在攻击完成之前就捕获到恶意行为。对于入侵检测设备而言，最基本的限制就是只能探测代码中要求其进行探测的行为。因此，不论系统的代码多完美，攻击者都可以找到绕开入侵系统的方法。安全管理员经常面临减少误判还是漏判两难的抉择。入侵检测系统可以采用如下所述的二分法进行分类：基于主机的系统和基于网络的系统，事后系统-实时系统-事前系统，以及错误行为/相似性探测系统-良好行为/偏差探测系统。

2.5.3 可信任部件的组合

为了降低大型系统的生产成本，需要尽可能多地使用现有的商业部件。对于 FPGA 器件而言，这就意味着需要复用现有的 IP。采用多个部件构建安全的嵌入系统会带来很多的难题。

2.5.3.1 组合问题

所有由于部件组合导致的问题中，最经典的例子就是所谓的级联问题[71]。这个问题描述如下：在一个 MLS 系统中，相关的标签通过比较运算符（≥）进行线性排序，如果在排序表中，两个标签之间没有其他标签，则说明这两个标签是相邻的。如果某个部件执行了充分的安全策略，将两个相邻的敏感性标签 S_i 和 S_j 所代表的信息分隔开，则此部件的保障度为 T_2。现在我们假设存在两个保障度为 T_2 的部件 C_1 和 C_2。假设组织的安全策略要求将三个相邻的敏感性标签 S_1、S_2、S_3 各自所代表的信息分开，则这种分离所要求的保障度为 T_3。如果采用 C_1 来分离 S_1 和 S_2，采用 C_2 来分离 S_2 和 S_3，则相关的体系结构可以认为具有足够的保障度。尽管如此，如果在后来将部件 C_1 和 C_2 在 S_2 级别处连接起来，如图 2-7 所示，则其组

合就构成了一个系统（一个虚拟部件）此系统跨越三个密级，但保障度只有 T_2。保障度不存在递增关系，因此如果网络安全策略要求将三个敏感性等级分开，使系统的保障度达到 T_3，则仅仅将两个保障度为 T_2 的部件串联起来是不够的。

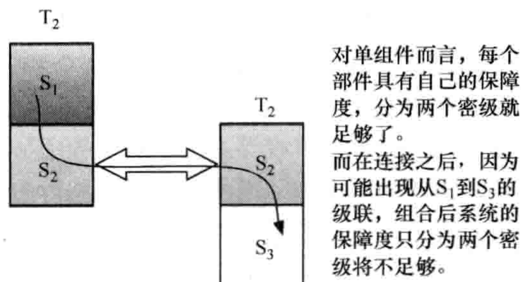


图 2-7　级联问题（单独的部件都具有足够的保障度，但将其组合起来之后，其保障度就再不能满足要求）

对级联问题的分析说明，在网络内识别级联情况的算法中涉及时间复杂度 $O(an^3)$ 以及一个空间复杂度 $O(an^2)$，其中 a 为涉及的密秘级别数量，n 为网络中的节点数量[45]。另外，计算一个校正的成本（即保留原策略对网络进行重构）是一个 NP 完全问题[45]。尽管网络重组所需要的成本为一次性投入成本，但是通过此分析得出的结论认为最好是一开始就避免在网络中出现级联现象。

避免产生这种组合问题的一种方法就是提供一个规则框架，在此框架内将经过预先分析的部件连接可以安全出现。例如，在 TCB 子集中[94]，将系统的安全策略进一步分割为一组监视器，其中每个监视器负责执行总体策略中的一个子集。例如，一个监视器执行强制性保密策略，另一个监视器执行完整性策略，第三个监控器执行非强制保密性策略。只有在所有三个监视器都允许的情况下，才会授予主体访问客体的权限。如果系统可以划分成包含监视器的组件，通过合理的工程化（包括严格的设计及接口要求组合）可以构建出一种其子集组件结合起来能够满足总体策略目标的体系结构。在这种情况下，目标在于能够构建出可对其进行独立评估的 TCB 子集，并保证组合后能够执行更大的系统策略[75]。这种开放方式获得的结果称为分区化 TCB。

> 设计提示：组合。保障度不具有递增性，在某些特殊情况下，甚至还具有递减性。两个在独立状态下保障度能够满足要求的部件，在将其连接起来之后所得系统的保障度可能无法满足要求。TCB 子集抽象需要将安全策略划分为一组执行机制，每个执行机制都执行总体策略中的一个子集。只有在通过所有执行机制的许可后，才能授予相关的访问权限。尽管如此，按照部件进行评估得到总体结果依然是个难题，因为部分非预期的行为仍可能漏出保障度管理范围，造成保障度不能满足需求。

2.6 保障度管理策略的执行

在考察产品生命周期内所采用的每一项技术是否安全可靠，以及如何进行有效管理的时候，可以发现软件与可重构硬件之间存在着许多相似性。

FPGA器件中包括一组执行特定功能的逻辑元件，同时FPGA器件的可编程性需要一种方法来明确逻辑，进而定义FPGA的行为。正如软件的行为通过程序语言编写的程序来进行定义一样，FPGA中逻辑的行为模式通过硬件描述语言（HDL）来进行定义。另外，常见的FPGA逻辑元件能够通过库的形式表示，以便组合起来执行更加复杂的功能。从这个角度看，FPGA可视为一种持续的静态加载的程序，适用于软件的常规分析，保障度保证措施也适用于FPGA。

和橄榄球队中的进攻线一样，保障度的重要性经常容易被忽视，但对于产品在整个生命周期内的维护而言，保障度的确是一个致命的因素。正如进攻线一直在干脏活累活，保证四分卫、跑卫以及接球员能够穿过场地并赢得赞美一样，持续、严格以及成功应用合理高保障度的实现方式，以保证相关产品成功开发的情况是很少受到关注的。相反的是，正如只有在四分卫被罚下场之后人们才会注意到进攻线一样，通常只有在发现产品中的缺陷之后，人们才会开始注意保障度以及配置管理措施。

但是，正如了解体育的人所知，进攻方的成功开始于进攻线能够持续在防守方的防线上找出破绽，保护传球并保证队伍能够前进。对于保障度以及配置管理流程而言，同样必须在具有鲁棒性和高质量的产品的整个生命周期内实施相关措施。

2.6.1 生命周期支持

生命周期管理是产品开发及维护政策中不可或缺的一部分，它能够是帮助定义产品的保障度，和保证生产商的核心竞争力。定义合理有效的生命周期管理模型是保证大型复杂软件项目设计安全性的基石。不同组织机构定义的实际生命周期流程可能各不相同，这主要由产品的性质（硬件还是软件），以及要求的保护等级（高保障度还是低保障度）决定。无论如何，在相关产品的整个生命周期内，都必须实施相关的安全流程，包括要求的工程化、设计、开发、加工、测试、发售、维修以及寿命末期处理等阶段[108]。尽管工程化阶段并不受关注，甚至经常被忽视，但此阶段是保证安全性的一个重要方面。在此阶段中，必须对功能性以及保障度（非功能性）安全要求进行正确的定义，以避免错误的功能或者以错误的方式保护了正确的功能。

2.6.1.1 评估标准

通用标准（Common Criteria，CC）是一套国际公认的评估框架，它注重工程

化阶段的基本方面，同时推导出安全需求的方法学。该方法学的重点放在最终产品需要规避的、现实的和可能的威胁[23]上。由于通用标准中很多要求是针对生命周期建模、配置管理（CM）、安全交付、开发安全性以及缺陷修复的[25]，所以通用标准（CC）评估范例中，对生命周期的支持具有非常重要的作用。在通用标准的补充文件中对涉及可编程集成电路（例如，FPGA）的生命周期安全性问题，做了进一步的说明[22]。此文件中给出了将基本的通用标准（CC）评估方法学应用于必须根据通用标准要求进行评估的硬件 IC 产品的指导原则。在美国，根据第 11 号国家安全电信以及信息系统安全政策（National Security Telecommunication and Information Systems Security Policy）的规定[20]，所有国家安全系统都必须根据通用标准（CC）或者国家标准与技术研究所（NIST）的联邦信息处理标准（FIPS）的要求进行评估。这些系统中通常都带有可编程电路。

对于用于敏感但非保密环境的商用软件加密模块，FIPS 出版物中的 140-2[104]是目前类似于生命周期要求的正式评估标准（尽管其被误称为设计保证）。例如，其中的配置管理、安全交付和安装、开发证明以及操作规程等都与生命周期类似。FIPS 第 140-2 号文件中定义了四种层次等级，同时明确引用了通用标准（CC）作为在目标加密模块中使用的软件的安全要求。在当前的 FIPS 第 140-3 号文件的草案中[107]，删除了文件与通用标准之间的引用关系。此文件从 2007 年 7 月开始进入公开征求意见阶段。在草案文件中，设计保证被重新命名为生命周期保证，还添加了必须使用自动配置管理系统、必须进行供货商测试、以及更加严格的开发流程等要求。例如，安全等级在 2 级及以上的自定义集成电路必须使用高层次硬件描述语言[107]进行设计。

2.6.1.2 可信任工具的使用

FIPS 第 140-3 号文件的草案中同时还要求，如果软件中包括有加密模块，即使相关软件具有最低的安全等级（1 级），也必须提供编译器、配置设置以及用于产生可执行代码方法的相关信息。这涉及令人烦恼的可信任工具问题，即用户如何确定用于产生可执行代码或加工硬件电路的工具的正确性。对于 FPGA，由于在 FPGA 产品在设计、加工、组装、测试及发售过程中使用的工具更加复杂，因此问题更加严重。这些工具通常都是由不同的供货商（包括国内供货商及国外供货商）提供的，没有标准化的计量方式或标准来评估工具实现的完整性。从理论上讲，对所有工具进行形式验证可以在很大程度上保证最终产品不会被相关工具植入后门。但在实践中，这种做法会使成本增加到无法接受的程度。有关人员已经进行了关于这些挑战及其他与集成电路（包括 ASIC 以及 FPGA）保障度相关问题的调查，并撰写了题为《Defense Science Board Study on High-Performance Microchip Supply》的研究报告[113]。这个报告促使国防高级研究计划局（DARPA）在 2007 年发布“集成电路保障度”研究邀请[114]，致力于开发不论相关设计或者加工过程在何处进行，

都能够对 IC 硬件产品以及其设计进行严格验证的技术。在一篇题为《The hunt for kill Switch》的文章中[1]，专门对特洛伊木马进行了讨论，得到的结论是：目前高度复杂的硬件系统天生就带有无数的缺陷及漏洞。

对于安全的软件开发而言，最佳的实践方式应包括以下步骤：①基于质量因素（如来源、成熟度、稳定性及广泛应用程度）等进行通常的经验分析，仔细选择工具；②针对工具的功能接口进行彻底的黑盒测试及安全性分析；③采用严格的配置控制措施维护相关工具。此流程如果能够正确地实施，就能有效地规避恶意后门，以及无意误用导致的威胁。测试及测试结果分析可提供相关证据，以确认所选择的工具中不存在恶意功能。本书第 8 章讨论了如何将该理念应用到未来的 FPGA 设计工作中，以及如何使用类似的流程来选择和管理 FPGA 设计流程中不同阶段（例如，逻辑综合、布局和布线等）所使用的工具。

2.6.1.3 在生命周期内应用安全原则

作为深度防御策略的一部分，有效的生命周期管理方法学中应该包括如下高保障度软件的安全原则：

1）审查；

2）最小权限；

3）职责分离。

审查是以问责为目的持续检查和评估的纪律。为了发现安全违规事件、威慑渗透的意图，在审查框架中应该同时包括自动技术措施以及手动操作。应用于生命周期管理时，审查可以保证所有设计及加工工作都符合生命周期控制策略和流程要求。这将能够有效弥补由恶意攻击者的安全违规事件导致的不良影响。威慑是一种有效的风险管理机制，要求在系统生命周期内的所有阶段，同时执行随机和定期审查的策略，这同样可以使恶意攻击者气馁，放弃发动攻击的意图。配置管理（将在下文详细讨论）就是审查的一种形式。这种机制能够识别对系统的保障度造成影响的开发及运行过程中的配置变化情况。

在论述信息保护的开创性著作中[85]，Saltzer 和 Schroeder 提出了遵守诸如最小权限以及职责分离等安全原则，坚持这些原则还能够提供保护并控制损害。他们将最小权限定义为一种设计约束，遵守该约束可将由程序或操作错误所导致的损失控制在一定的范围之内，同时将权限（即职责）分离定义为能够降低被串通或被恶意实体泄露风险的措施。FPGA 设计以及加工流程是一个由不同活动组成的复杂流程，涉及许多不同要素，各个要素之间相互依赖，确认不同部件的实现（例如网表）没有违背设计的原始意图（例如，HDL 设计文件）是非常艰巨的工作。使用加密技术能够保护此设计流程的部分阶段[102]。在设计流程中，遵守最小权限原则以及权限分离原则（例如，隔离关键步骤、执行不同角色、将权限只局限于手头的任务等）能够有效地增加最终产品的保障度。

2.6.2 配置管理

技术在不断地进步，因此变化是不可避免的。配置管理（CM）是一种行之有效的变化控制的手段，无论是硬件产品还是软件产品，都应该在生命周期的早期实施配置管理。改装配置管理（指发现问题后再添加配置管理）的成本昂贵，并有可能对产品的完整性造成严重影响，因为保持完整的未经篡改的变更档案是实现追溯的目的，是配置管理的基石。配置管理可以视为主动防御机制，在正确实施配置管理的情况下，能够有效规避与配置变化相关的固有安全风险。

大部分开发人员都知道配置管理，很多人都将其视为版本控制机制。很多开发人员不是忽视就是低估配置管理的重要性，他们担心配置管理的控制和步骤会加重设计负担。但配置管理是一种严格的安全保护措施，正确地实施配置管理能够有效地确定硬件和软件组件的初始基准及组件后续的变化情况。变化控制在配置管理中至关重要，因为其主要目标是防止基准配置项在未获得授权的情况下被变更（包括意外修改）。在高保障度软件的开发过程中，需要对其进行的变更（例如，对现有组件的修改以及添加新的组件）进行彻底的安全分析，以评估其对系统中其他部分安全性所造成的影响，必须在批准相关变更之前执行和审查这些安全分析。同时还必须采用系统的制衡措施来威慑串通行为（例如，严格区分配置管理与开发环境），确保安全分析的有效性（例如，确保安全分析是由经过培训的安全分析员执行，审查由变更控制委员会执行）。在 FPGA 的开发过程中，配置管理目标和要求同样适用，尤其对由多人（核设计师、系统开发者以及生产商等）开发和维护其代码的复杂 FPGA 系统更是如此。

配置变更的控制非常重要，但仅采取这种措施是不够的。NIST 定义了一组建立配置管理所需要的广泛的配置管理要求（例如，安全性控制），从建立配置管理策略到系统中所使用组件当前目录的维护[106]。需要根据安全事故对整个系统所造成的潜在影响，针对不同的信息系统采用上述配置管理要求中的不同组合。根据安全损害事件对组织机构业务及资产影响的严重程度，FIPS 第 199 号文件中定义了三个不同程度的影响级别。影响程度有限的为低级，影响严重的为中级，影响非常严重或者灾难性的为高级[105]。NIST 编号为 800-53 的专刊中对影响程度为中级及高级的系统提出了下述八项配置管理要求[106]：

1）配置管理策略及流程；
2）基线分配；
3）配置变更控制；
4）监视配置变更；
5）变更访问限制；
6）配置设置；

7）最少的功能；

8）信息系统组件清单。

这些要求涵盖了芯片产品生命周期中的所有阶段，包括策划、开发以及部署阶段。同时它们也适用于系统中的所有组件，包括 FPGA。换而言之，除了对单个产品的开发的保障度具有效果之外，在系统的安全策略及计划中使用时，还能够提高系统的任务保障度。系统的安全状态部分取决于经过认可的配置设置，如果在未进行合理的分析以及追踪的情况下对其进行修改，则可能会导致系统的认证（例如，使用授权）失效。此原则同样适用于 FPGA 嵌入系统，在现场改变比特流文件可能会同时对系统的性能以及安全性造成负面影响。因此，应该建立并执行配置管理策略以及流程，以降低与比特流重配相关的风险。

2.6.3　独立评估

问责制和透明度是构建和维持保障度的核心要素。Juvenal 关于信任的深刻观察㊀多年来一直被用于强调必须进行外部监督以加强管理问责制。在安全产品的开发过程中，很容易就能找到透明度需求和安全评估需求之间的平衡点，因为若对产品曾做过独立的安全性评估，则产品的安全档次就能提高。

为了确保产品的保障度是可信的，产品的安全评估应该由客观的第三方独立进行，这个第三方最好是获得了政府批准的机构。这主要是由于公正和独立的第三方对于防范偏见及串通阴谋是必不可少的。总而言之，若供货商能够声明其产品已经通过承担法律责任的官方机构的认证，则用户对其产品的信任度将会增加。例如，在生命危险的情况下，如果医生处方中的药物曾获得美国食品及药品监督局的批准，则美国医生和患者会感觉更安全。同样，对安全非常重视的 IT 用户也倾向于使用已通过诸如国家信息保障合作组织（NIAP）、评估及认证体系（CCEVS），以及美国国家标准技术研究所（NIST）验证的安全产品。

评估及认证体系（CCEVS）要求由美国国家标准技术研究所（NIST）认可的商业测试实验室，按照 CCEVS 体系，对安全产品的评估过程，进行全面的监督并予以认可[12]。这个测试实验室根据安全保障评估及认证的标准过程，以及 NIAP 认可的保护概要，对相关安全产品进行安全评估[13]。评估及认证体系（CCEVS）是一个国际标准，每个国家都有类似于 CCEVS 但又具有自己特色的安全保障评估及认证体系。在美国，加密模块的认证由 NIST 而不是由 CCEVS 负责。NIST 与加拿大政府通信安全部门协作，根据 FIPS 的加密标准（例如，FIPS 140-2 标准），对加密模块验证计划（Cryptographic Module ValidationProgram）进行监管[103]。

诸如 CCEVS 和 NIST 等产品评估机构只负责评估单个产品（例如，操作系

㊀ “Quiscustodietipsoscustodies”（“谁来监督监督者呢?”）——Juvenal，讽刺文学 VI. 347。

统、防火墙、网站服务器等）的保障度，而不负责评估使用由已通过评估产品所构建的最终系统的保障度。从系统采购的角度来看，针对单个产品进行的独立安全评估是技术尽责调查中非常关键部分，技术尽责调查工作做得完善，能够有效地规避系统生命周期内的相关风险。尽管如此，由已经评估的多个不同产品组成的系统并不一定安全，这主要由于经过评估产品的安全功能相互作用，可能会导致新的缺陷或者漏洞。产品评估是基于安全假设（例如，物理安全以及人身安全）以及产品预计使用的具体操作环境中可能出现的威胁进行的。在将不同的威胁模型集成到最终系统中时，已经评估的保护机制可能不足以规避在系统层面所面临的威胁。

在实现的不同层面上（例如，硬件层、操作系统层、应用程序层）对集成保护机制进行独立检验能够在将由部件构成的系统部署使用前，识别恶意行为。根据美国联邦政府规定，联邦机构在对系统提供操作授权之前用来进行安全及风险评估的过程被称为认证及鉴定（C&A）。在获得授权的系统或者其操作环境发生变化时，根据组织认证及鉴定政策，可能需要进行后续的认证及鉴定，以判定并规避由于变化导致的风险。

尽管产品中的 FPGA 设计的复杂度通常都是隐藏（封装）在较高级别的功能部件（例如，处理器核和设备控制器）中，非常重要的一点是在总体系统安全评估过程中不能忽视 FPGA 的扩展性。应该对 FPGA 的使用进行审查，审查的深度及严格程度应与产品中关键软件安全性审查程度相同。动态可重构是使用 FPGA 的主要好处，但这也是一把双刃剑。在将基于 FPGA 的组件用于关键系统时，较为谨慎的做法是将其作为系统监督及认证流程的一部分，进行系统结构及设计分析，以发现由于不好、不正确或未按照规定使用该 FPGA 组件导致的副作用。

2.6.4 动态程序分析

动态程序分析通常是指程序执行过程中的测试和分析。首先向被测试的目标程序输入专用的测试矢量组合，然后采用设备对程序的行为进行检查和确认。用构造的测试矢量作为被测试程序的输入激励，用测试设备做如下四项检查：

1）功能测试；

2）性能测试；

3）时序约束测试；

4）资源利用率测试。

功能测试是动态程序分析中最常见的测试方式。在功能测试中，向被测程序输入一组专门设计的测试矢量，检查每个接口输出的结果是否正确、有否出现错误及异常，以验证被测试程序是否符合设计要求。功能测试通常和代码覆盖分析一起进行，以保证所有程序代码都被测试矢量调用和执行过。根据代码覆盖率编写测试矢

量集，而且产生特定响应的测试矢量集也需要专门设计。

在很多情况下，只对程序进行功能接口的全面测试就足够了，因为用户对于程序的其他属性（例如，性能、时序约束、资源利用率等）没有特殊的要求，或者根本就没有要求。但在另一个极端，对于嵌入式实时系统而言，资源以及时序限制对于程序的正确执行至关重要。通常情况下，很难预先了解某个程序在执行过程中的性能或者资源利用率情况。在这种情况下，应用动态程序分析可以确保系统能在有瑕疵的现实正常状态下正确地运行。

2.6.4.1　测试

测试发生在软件开发及认证的不同阶段和不同场合。单元测试和部件测试通常在开发阶段由软件开发人员完成。对于不同的组织机构而言，由开发者实施的测试的严格程度也大不相同。

在完整的软件产品被开发完成后，必须对其进行系统测试，系统测试一般是由专门的质量保证小组负责进行。测试要求必须根据完整描述产品接口的规格说明书编写。此层面通常采用的测试流程可以归纳为

1）对程序的所有接口进行 100% 的调用。

2）对所有外部可见的条件进行行为验证。

3）需要同时测试成功条件以及不成功条件，包括使程序产生的所有错误及异常。

如果相关产品还需要进行认证，则评估人员还要添加一些测试。随着认证的档次不同，产品的检验及测试流程也大不相同[26,109]。上述测试仅仅由只针对相关的产品运行的一个简单测试套件组成，用以确认产品的功能能否满足要求。尽管如此，通常情况下，认证过程需要对产品的符合性进行评估，不只是针对产品某个时刻的符合性，还应涵盖产品的整个生命周期。根据这种类型的要求，在认证过程中，不但要检查产品本身，同时还需要对产品开发及维护过程中应用的所有软件的开发过程进行检查。通常情况下，评估人员无法对由产品开发者进行的所有测试进行检查，更无法重复所有测试。因此，评估人员只能对测试的流程、测试记录、文件材料以及其他能够证明软件在其生命周期内的保障度的材料进行检查和分析，并以此来代替实际的测试。

尽管在软件开发过程中，测试是非常重要的一个阶段。但在整个开发周期内，测试通常是分散进行的，通常在某个软件组件，甚至完整的软件程序编码完成之后才进行测试。

在整个软件开发周期中考虑软件测试需求能够大大提高软件产品的质量，这不仅要求在软件开发的合理时间点进行测试，还要求在程序设计和开发过程中主动地考虑测试的作用。面向测试设计策略中不仅包括使所构建的接口符合简洁性、适用性的设计原则，还包括应用有助于测试的编程技术。

应用设计原则也就是说编写完善的抽象模型并构建合适的模块化架构，可以使程序变得更简洁直观，因此也更便于测试。有意思的是，虽然设计原则中的抽象模型和合适模块不易掌握，但是即使生搬硬套这些原则，也能得到很有价值的回报。例如，某接口被认为是一个测试案例非常多的复杂接口，很难测试。这意味着假如我们能理解该接口的特殊功能需求，则该接口或许不必设计得那么复杂。

采用易于测试的设计技术使我们能够开发出大量的组件块和配套的测试模块，这些组件块和配套的测试模块既可以用于组件块初始开发阶段的测试，也能用于系统综合时的回归测试。组件块的测试通常使用测试激励模块进行。这些测试激励模块在测试过程中不仅调用软件模块的外部接口，同时还需要利用代码让调试人员能够有选择地暴露并控制软件模块的内部状态。

与传统软件开发流程类似，FPGA 开发过程中也涉及到迭代步骤。在这些步骤中，需要将某个开发步骤中的某个输出结果反馈到电路模块中，以便验证电路的功能。为了支持这个迭代模型，FPGA 开发工具已具有支持硬件描述语言的复杂测试技术，这些测试技术能够在开发过程的每个阶段构建测试矢量，对由 HDL 描述的电路及接口进行测试。

2.6.5 可信任发售

对可信任产品的交付过程必须进行保护，防止产品从供货商送到用户的过程中被篡改或者被植入后门，这是非常重要的。对于所接收到的产品，用户必须得到下述保证：

1）产品版本是供货商明确的正确的版本。如果相关产品已经过评估，则发售的版本必须和已经评估的版本完全一致。

2）相关产品来自于供货商，而不是来自于可疑的货源。

3）到货时产品必须未被修改。

在评估标准中，这种类型的保障度要求通常分别称为可信任发售[110]和安全交付㊀[25]。提出这两个要求的主要原因在于，如果在其生命周期中相关产品的安全机制在未获得授权的情况下被修改，则会对其执行安全策略的能力造成负面影响。在开发阶段，通过配置管理能够防止产品被恶意修改，可信任发售能够消除产品发售阶段被篡改或者被植入后门的威胁。

在可信任计算机系统评估标准（TCSEC）中，只对 A1 级别的产品提出可信任发售要求。这主要是由于对于保障度等级较低的产品而言，提供安全交付保证的措施成本相对较高[111]。尤其是 TCSEC 要求供货商实施能保证交付后产品完整性的发

㊀ 在本节内容中，这两个术语具有同等的含义。

售系统，同时应该向用户提供对接收到的产品进行验证的流程，以确定是否和供货商所发货的版本一致[110]。这些要求适用于产品的初次交付，也适用于后续升级包的交付。与 TCSEC 一起的还有许多技术指导原则，这些技术指导原则的目的在于进一步澄清 TCSEC 要求，并同时提供具体实施方法。其中一份文件叫做《A Guide to Understanding Trusted Distribution in Trusted Systems》（可信任发售以及可信任系统理解指导原则）[111]。在这种指导原则中，解释了可信任发售对于产品生命周期保障度的重要性，同时提供了数种实施有效的可信任发售机制的方法。

另一方面，通用标准要求从评估保证等级为 2 级（EAL2）（七个评估等级中的第二低的等级）开始的产品都应进行可信任发售。在之前版本的通用标准（2.3 版及更早的版本）中，可信任发售被要求组合成一个系列（ADO _ DEL)，同时使用交付流程（EAL2 和 EAL3)、篡改检测（EAL4 ~ EAL6)，以及篡改预防（EAL7）等术语进行描述 [21]。这些分类间具有线性等级关系，即篡改检测同时要求交付流程，篡改预防同时要求篡改检测及交付流程。在以下方面，通用标准的要求类似于 TCSEC，换言而之，这两份标准都要求使用供货商原版版本，同时在交付渠道的两端都采取流程以及技术措施。

在当前的 3.1 版的通用标准中，可信任发售要求采用交付流程和准备流程这两个术语进行描述[25]。前一个术语要求供货商在产品的发售过程中记录并使用相关的交付流程。后一个术语要求供货商在用户现场为用户提供充分的验收流程。与 2.3 版相比，通用标准中的保证要求进行了大规模的修订，其中可信任发售要求定义为两个单独的分类（ALC-DEL 和 AGD _ PRE)，使得通用标准的新用户更加难以遵循。之前版本中明确提出的防篡改和防伪造要求，在新的通用标准版本中被移到了应用笔记中，不再是通用标准的规范。

尽管 TCSEC 和通用标准中关于可信任发售的要求的适用范围以及形式都不相同，但两者都有着相同的目标，即防止软件产品在开发完成后被篡改或者被盗用。FPGA 产品也存在类似的威胁，正如高保障度软件一样，对于 FPGA 产品也应该设置严格的交付机制，以规避基于 FPGA 的产品在不同开发阶段中可能面临的威胁。

2.6.6 可信任恢复

用状态机模型实现的安全系统必须保证初始的安全状态在每次状态转换之后都能够转换成另一个安全状态[8]。尽管随着系统安全策略模型的不同，对安全状态的定义也有所不同。总而言之，安全状态是指系统数据始终如一、不会被破坏，同时系统能够正确地执行由其安全模式确定的安全策略的系统状态[46]。

当系统检测到不再处于安全状态时，系统必须努力尝试自动恢复到安全状态，同时保证在恢复过程中不需要再增加保护措施，就能进入安全状态。在这种有连续性保护的前提下，出现异常情况时系统能自行恢复的概念被称为可信任恢复。TC-

SEC 和通用标准（CC）进一步将这些异常情况分类为故障或者运行中断。故障可以是系统安全功能㊀中的一个错误状态，该错误状态会导致系统行为的不正常（例如，由于硬件的暂时故障导致系统数据结构中的数值不正常）或者媒介故障（例如，磁盘损坏）。另一方面，运行中断指的是由于不正确的人工干预导致的错误，例如系统非正常关机导致的错误[24,112]。

当系统处于运行状态（与挂起状态相对）时，可能处于两种模式中，即运行模式或者维护模式。这两种模式都必须支持可信任恢复机制[46]。这些机制必须能够确定系统当前状态是否安全，同时在系统处于不安全状态时，必须启动针对具体模式的自动机制（例如，磁盘坏道的重新扫描）来修复系统。某些错误状况可以通过自动机制修复（例如：重新映射磁盘坏扇区），而另一些必须通过手动恢复（例如，由于意料之外的错误导致系统崩溃）。根据系统的具体操作环境，可以采用各种不同的恢复方法。例如，与传统计算机相比，在嵌入式实时系统中采用的恢复方法可能更加复杂，这主要是由嵌入系统资源的限制导致的（例如，处理器消耗以及响应时间）[62]。

尽管自检测是保证系统完整性的要求，自检测也与可信任恢复相关。在系统初始化以及正常操作过程中，通过自检测可以发现需要恢复的不正常状况。另外，作为恢复流程的一部分，自检测还可以被自动恢复机制启动或者由管理员启动，以确认系统恢复完成之后是否处于安全状态。

至于确保生命周期内的保障度，系统采用的恢复机制必须能达到与其他安全有关功能相同的保障度开发要求，因为恢复机制也是系统安全功能的一部分。其设计以及实施必须经过仔细分析和审查，以确保支撑体系结构特征（例如，自我保护、最小权限、模块化及最小化等）。同时还要确保代码中没有特洛伊木马或者后门。另外，必须进行安全测试以及漏洞分析以确定在恢复过程中可以被利用来绕开安全机制潜在的安全漏洞[26]。

从具体的故障中恢复可能需要复杂的管理操作。与普通用户相比，管理员用户通常具有更多的特权，因此最小权限原则也应该适用于系统恢复权限的分配过程，以确保只有获得相关授权的管理员用户（即安全管理员而不是系统操作员）才可以执行恢复操作。在操作指导文件中必须详细说明所有类型的故障条件、恢复流程以及所用工具。同时对于每种类型的故障，应该提供具体的指导原则，说明应该采用何种方式才能够最好地从故障中恢复。最重要的是关于恢复的用户文件必须完整、正确，否则，误用恢复功能可能会对系统的能力以及安全性造成影响，损害系统的保护。

㊀ 安全功能性这个术语的含义和 TOE 安全功能（TSF）的含义相同，此术语在通用标准中定义为必须依赖安全功能要求得到正确执行的所有 TOE 硬件、软件以及固件的组合[23]。

2.6.7　静态分析

与动态程序属性的测试不同㊀，静态分析只对有关程序的说明书做客观的审查，然后再确认该程序的属性。

根据抽象程度的不同，可对程序进行分类，例如，用户手册、设计规范、源代码及可执行代码是抽象层次各不相同的程序。因此可采用不同形式的静态分析对不同抽象层次的程序进行检查。通常情况下，静态分析这一个术语专指源代码分析，但在本书中，其含义更加广泛[17]。

2.6.7.1　代码分析和错误检查

静态分析的基本形式就是代码审查。在代码审查过程中，程序作者将与同事、设计师、管理人员及客户等一起查阅程序代码。代码审查的重点是关注程序的属性（诸如编程风格是否优雅、程序是否符合高层设计说明书的要求），也关注程序中的错误（诸如缓存溢出等）。只要程序的有关属性可以用有效程序（例如，规则）这个术语来定义，它就能够被相关工具理解，则自动静态分析工具[6,15,16,28]就能对说明书中的这部分规范进行分析。尽管分析工具能够搜索并发现定义清晰的缓存溢出情况，但诸如程序风格等很多概念相对较为主观，而且超过了当前分析工具的能力范围。可以通过自动分析工具进行静态分析的源代码属性包括语法以及格式的正确性、内存泄露、堆栈不正确或者通用内存访问、内存泄露、权限过度使用、缓存溢出、包含无用或者重复的功能、无用变量、无初值变量缺乏封装/数据隐藏以及检查时间和使用时间错误等。

与自动测试一样，自动代码审查及自动错误查找并不能保证相关属性一定能在给定的程序中正确完整地实现（编写一个能够理解其他程序的程序，在理论上存在的困难就是所谓挂起问题[11]）。使用下一节中阐述的形式化方法能进一步提升程序的安全性。

2.6.7.2　形式化方法

人类语言的发展趋势是趋于模糊和缺乏精确性，而数学为清晰而精确的描述以及对描述进行推理提供了基础。形式化方法本身并非是当前计算机科学中的一个正式定义或者标准的术语，而通常是指在软件以及硬件系统开发过程中的许多方面使用数学方法。尤其是对于高保障度或者高鲁棒性等级的产品而言，需要采用形式化方法来验证其安全性[14]。

部分形式化方法包括以下内容：

1）通用数学模型：

①计算及处理[44]；

㊀　这里所述的程序可以是模块、组件、单片机系统或者分布式系统。

②安全性[8,9]。

2）能够表述下述内容的具有精确含义的规范语言[84,97,98]：

①系统行为；

②安全性能；

③不同规范之间的从属关系；

④性能一致性定理；

⑤更多抽象规范一致性定理。

3）具有精确含义的可执行安全语言：

①用一阶语言结构可对安全属性（诸如正确的 MLS 流等）进行描述；

②程序成功地通过编译，说明该程序所包含的属性符合语法要求[30,115]。

4）能够操纵形式规范逻辑的自动系统，例如：

①自动或者交互性定理证明器[54,78,84]；

②模块查找器，模块执行器，以及 SAT 解算器[48]；

③能够根据形式规范特性自动生成定理的工具[34]。

5）信息流分析工具。

2.6.7.3 属性的细节化和保护

安全系统的形式验证包括不同抽象层次上关键技术规范的形式化，还包括一系列相应的范例以表明每个规范中都保留有下一个最抽象层的安全属性-这产生参数的传递，从而使系统的安全策略（见图 2-8）得以实施。此链上的形式化要素越多，结果参数的形式化程度就越高。将某特定规范转化为抽象程度更低的规范细节的过程称为细节化，相反将具体的规范转化为更加一般和广义的规范的过程称为抽象化。

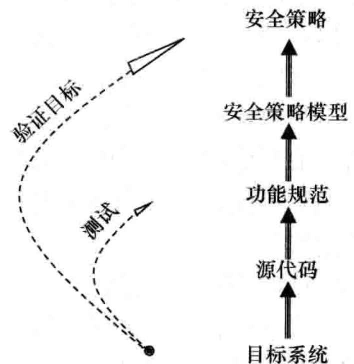


图 2-8 形式验证的证据链

高保障度验证标准[14,110]通常必须符合以下各项要求：包括安全策略模型以及顶层功能规格说明（包括输入、输出、处理以及每个接口的内部实现）在内的形式化规范；表明此形式化模型符合其安全特性的证据；表明形式化规范中保留模型有关属性的证据。形式化验证的方法可用于隐蔽信道分析，也可用于演示源代码与形式化顶层规格说明完全一致。

通常我们希望采用自然语言表述的安全策略和安全策略模型都十分简单，只要通过查阅程序就有足够的把握确保安全策略与所编写的程序完全一致。安全策略模型通常是安全策略的具体化，其中的组织机构级策略，在计算机技术领域内，被认为是“不需要考虑使用计算机的”[110]。换而言之，安全策略模型有助于安全策略与形式化规范之间的转换。另一方面，源代码到机器代码的可信任翻译（例如，

可信任编译器）和形式化功能规范自动翻译为源代码或者机器代码是当前研究的热门主题。

在所谓的细节化悖论的问题中[70, 82]，已被证明信息流模型[39]的细节化过程通常不能保持模型的安全属性。例如，细节的添加（即在形式化规范中）可能会引入未包括在抽象形式模型中的信息流。在这种情况下，若执行隐蔽信道分析，则可以确保在已细化的规范中的信息流依然正确。同样，若使用访问控制模型[8]来执行形式化规范的隐蔽信道分析，则可确保模型中外来的信息流不违反安全策略。

源代码和形式化规范之间的正确对应关系可以通过详尽地枚举源代码加以表述，在此过程中，应该将代码中的每个元素都映射到其形式化规范中的代表项，同时分析解释形式化规范语义保留在每个细节中的原因。研究从形式化规范自动转换成源代码的编译工具，其目的之一是编程自动化，避免人工编写代码，以降低人工编写代码的错误概率。

参考文献

1. S. Adee, The hunt for the kill switch. IEEE Spectrum **45**(5), 34–39 (2008)
2. P. Ammann, R.S. Sandhu, The extended schematic protection model. J. Comput. Secur. **1**(3, 4), 335–385 (1992)
3. J.P. Anderson, Computer security technology planning study. Tech. Rep. ESD-TR-73-51, Air Force Electronic Systems Division, Hanscom AFB, Bedford, MA, 1972. Also available as vol. I, DITCAD-758206. Vol. II, DITCAD-772806
4. E.A. Anderson, C.E. Irvine, R.R. Schell, Subversion as a threat in information warfare. J. Inf. Warfare **3**(2), 52–65 (2004)
5. M.J. Bach, *The Design of the UNIX Operating System* (Prentice Hall, Inc., Englewood Cliffs, 1986)
6. T. Ball, E. Bounimova, B. Cook, V. Levin, J. Lichtenberg, C. McGarvey, B. Ondrusek, S.K.R. Jamani, A. Ustuner, Thorough static analysis of device drivers. SIGOPS Oper. Syst. Rev. **40**(4), 73–85 (2006)
7. D.E. Bell, L. LaPadula, Secure computer system: unified exposition and multics interpretation. Tech. Rep. ESD-TR-75-306, MITRE Corp., Hanscom AFB, MA, 1975
8. D.E. Bell, L. LaPadula, Secure computer systems: mathematical foundations and model. Tech. Rep. M74-244, MITRE Corp., Bedford, MA, 1973
9. K.J. Biba, Integrity considerations for secure computer systems. Tech. Rep. ESD-TR-76-372, MITRE Corp., 1977
10. E.W. Bobert, On the inability of an unmodified capability machine to enforce the *-property, in *Proceedings DoD/NBS Computer Security Conference*, September 1984, pp. 291–293
11. G. Boolos, R. Jeffrey, *Computability and Logic* (Cambridge University Press, Cambridge, 1974)
12. CCEVS, Publication #4: guidance to CCEVS approved Common Criteria testing laboratories, version 2.0. National Information Assurance Partnership Common Criteria Evaluation and Validation Scheme, September 2008
13. CCEVS, Publication #1: organization, management and concept of operations, version 2.0. National Information Assurance Partnership Common Criteria Evaluation and Validation Scheme, September 2008
14. CCMB, Common Criteria for information technology security evaluation, revision 3.1, revision 1, no. CCMB-2006-09-001. Common Criteria Maintenance Board, September 2006
15. B.E. Chelf, S.A. Hallem, A.C. Chou, Systems and methods for performing static analysis on

source code. US Patent 7,340,726, Coverity, Inc., 2008
16. H. Chen, D. Wagner, MOPS: an infrastructure for examining security properties of software, in *Proc. 9th ACM Conf. Computer and Communications Security (CCS 02)*
17. B. Chess, G. McGraw, Static analysis for security. IEEE Secur. Priv. **2**, 76–79 (2004)
18. S. Christy, R.A. Martin, Vulnerability type distributions in CVE. http://cve.mitre.org/docs/vuln-trends/index.html, May 2007
19. J.P.A. Co, Computer security threat monitoring and surveillance. Tech. Rep., James P. Anderson Co., Fort Washington, PA 19034, February 1980
20. Committee on National Security Systems, NSTISSP no. 11, revised fact sheet. National Information Assurance Acquisition Policy, July 2003
21. Common Criteria Maintenance Board, Common Criteria for information technology security evaluation, part 3: security assurance components, version 2.3, CCMB-2005-08-003. Common Criteria Maintenance Board, August 2005
22. Common Criteria Development Board, The application of CC to integrated circuits, version 2.0, revision 1, CCDB-2006-04-003. Supporting document, mandatory technical document. Common Criteria Development Board, April 2006
23. Common Criteria Maintenance Board, Common Criteria for information technology security evaluation, part 1: introduction and general model, version 3.1, revision 1, CCMB-2006-09-001. Common Criteria Maintenance Board, September 2006
24. Common Criteria Maintenance Board, Common Criteria for information technology security evaluation, part 2: security functional components, version 3.1, revision 2, CCMB-2007-09-002. Common Criteria Maintenance Board, September 2007
25. Common Criteria Maintenance Board, Common Criteria for information technology security evaluation, part 3: security assurance components, version 3.1, revision 2, CCMB-2007-09-003. Common Criteria Maintenance Board, September 2007
26. Common Criteria Maintenance Board, Common Criteria for information technology security evaluation, evaluation methodology, version 3.1, revision 2, CCMB-2007-09-004. Common Criteria Maintenance Board, September 2007
27. M.A. Cusumano, Who is liable for bugs and security flaws in software? Commun. ACM **47**, 25–27 (2004)
28. M. Das, S. Lerner, M. Seigle, ESP: path-sensitive program verification in polynomial time, in PLDI 02: Programming Language Design and Implementation, June 2002, pp. 57–68
29. P.J. Denning, Virtual memory. ACM Comput. Surv. **2**(3), 153–189 (1970)
30. D.E. Denning, A lattice model of secure information flow. Commun. ACM **19**(5), 236–243 (1976)
31. D.E. Denning, An intrusion-detection model. IEEE Trans. Softw. Eng. **13**, 222–232 (1987)
32. J.B. Dennis, E.C.V. Horn, Programming semantics for multiprogrammed computations. Commun. ACM **9**(3), 143–155 (1966)
33. DigitalNet Government Solutions, Security target version 1.7 for XTS-6.0.E, March 2004
34. P. Eggert, D. Cooper, S. Eckmann, J. Gingerich, S. Holtsberg, N. Kelem, R. Martin, FDM user guide. No. TM-8486/000/04, Reston, VA: Unisys Corporation, June 1992
35. European Commission, Biometrics at the frontiers: assessing the impact on society. Tech. Rep., European Commission Joint Research Center (DG JRC), Institute for Prospective Technological Studies, 2005
36. R. Fabry, Capability-based addressing. Commun. ACM **17**, 403–412 (1974)
37. R. Fitzgerald, trans. *Homer: The Odyssey* (Vintage, New York, 1961)
38. L.J. Fraim, Scomp: a solution to the multilevel security problem. Computer **16**, 26–34 (1983)
39. J. Goguen, J. Meseguer, Security policies and security models, in *Proc. of 1982 IEEE Symposium on Security and Privacy*, Oakland, CA (IEEE Comput. Soc., Los Alamitos, 1982), pp. 11–20
40. G.S. Graham, P.J. Denning, Protection—principles and practice, in *Proceedings of the Spring Joint Computer Conference*, May 1972, pp. 417–429
41. I. Hadzic, S. Udani, J. Smith. FPGA viruses, in *Proceedings of the Ninth International Workshop on Field-Programmable Logic and Applications (FPL'99)*, Glasgow, UK, August 1999
42. M. Harrison, W. Ruzzo, J. Ullman, Protection in operating systems. Commun. ACM **19**(8),

461–471 (1976)
43. J.L. Hennessy, D.A. Patterson, *Computer Architecture: A Quantitative Approach*, 4th edn. (Morgan Kaufmann, San Mateo, 2006)
44. C.A.R. Hoare, Communicating sequential processes. Commun. ACM **21**(8), 666–677 (1978)
45. J. Horton, R. Harland, E. Ashby, R.H. Cooper, W.F. Hyslop, B. Nickerson, W.M. Stewart, O. Ward, The cascade vulnerability problem, in *Proceedings IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May 1993, pp. 110–116
46. IAD (Information Assurance Directorate), US Government protection profile for separation kernels in environments requiring high robustness. National Information Assurance Partnership, version 1.03 edn., 29 June 2007
47. Intel, Intel 64 and IA32 architectures software developer's manual, vol. 3A: system programming guide, part 1. Intel Corporation, Denver, CO, 253668-022us edn., November 2006
48. D. Jackson, *Software Abstractions: Logic, Language, and Analysis* (MIT Press, Cambridge, 2006)
49. A.K. Jain, S. Pankanti, S. Prabhakar, L. Hong, A. Ross, J.L. Wayman, Biometrics: a grand challenge, in *Proceedings of the 17th International Conference on Pattern Recognition*, August 2004, pp. 935–942
50. M.J. Kaminskas, *Risk Assessment/Risk Management*. Building Design for Homeland Security, vol. 5. FEMA, Risk Management Series ed. (2007). http://www.fema.gov/library/viewRecord.do?id=1939
51. P.A. Karger, Improving security performance for capability systems. Ph.D. thesis, University of Cambridge, Cambridge, England, 1988
52. P. Karger, A.J. Herbert, An augmented capability architecture to support lattice security and traceability of access, in *Proceedings 1984 IEEE Symposium on Security and Privacy*, Oakland, CA (IEEE Comput. Soc., Los Alamitos, 1984), pp. 2–12
53. P.A. Karger, R.R. Schell, Multics security evaluation: vulnerability analysis. Tech. Rep. ESD-TR-74-193, vol. II, HQ Electronic Systems Division, Air Force Systems Command, Hanscom Field, Bedford, MA 01731, June 1974
54. M. Kaufmann, J. Moore, An industrial strength theorem prover for a logic based on common Lisp. IEEE Trans. Softw. Eng. **23**(4), 203–213 (1997)
55. G.H. Kim, E.H. Spafford, The design and implementation of Tripwire: a file system integrity checker, in *Proceedings of the 2nd ACM Conference on Computing and Communications Security (CCS)*, Fairfax, VA, November 1994
56. P. Kocher, Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems, in *Proceedings of the 16th Annual International Cryptology Conference (CRYPTO)*, Santa Barbara, CA, August 1996
57. M. Kurdziel, J. Fitton, Baseline requirements for government and military encryption algorithms, in *MILCOM*, vol. 2, Oct. 2002, pp. 1491–1497
58. L. Lack, Using the bootstrap concept to build an adaptable and compact subversion artifice. Master's thesis, Naval Postgraduate School, Monterey, CA, June 2003
59. B.W. Lampson, Protection, in *Proc. 5th Princeton Conf. on Information Sciences and Systems*, Princeton, NJ, 1971
60. B.W. Lampson, A note on the confinement problem. Commun. ACM **16**(10), 613–615 (1973)
61. C.E. Landwehr, Formal models for computer security. ACM Comput. Surv. **13**(3), 247–278 (1981)
62. K. Lee, L. Sha, Process resurrection: a fast recovery mechanism for real-time embedded systems, in *Proceedings of 11th IEEE Real Time and Embedded Technology and Applications Symposium 2005* (RTAS 2005), March 2005, pp. 292–301
63. T.E. Levin, C.E. Irvine, T.D. Nguyen, Least privilege in separation kernels, in *E-business and Telecommunication Networks; Third International Conference*, ed. by J. Filipe, M.S. Obaidat. ICETE 2006, Set'ubal, Portugal, 7–10 August 2006. Communications in Computer and Information Science, vol. 9 (Springer, Berlin, 2008)
64. T.E. Levin, C.E. Irvine, C. Weissman, T.D. Nguyen, Analysis of three multilevel security architectures, in *Proceedings 1st Computer Security Architecture Workshop*, Fairfax, VA,

November 2007, pp. 37–46

65. H.M. Levy, *Capability-based Computer Systems* (Digital Press, Bedford, 1984)
66. S. Lipner, The trustworthy computing security development lifecycle, in *Proceedings 20th Annual Computer Security Applications Conference* (IEEE Comput. Soc., Los Alamitos, 2004), pp. 2–13
67. Lockheed-Martin/The Open Group, Protection Profile for PKS in environments requiring high robustness. Draft Version 1.3, submittal for NSA approval, 09 June 2003. http://www.csds.uidaho.edu/pp/PKPP1_3.pdf. Last accessed: 15 March 2009
68. T.F. Lunt, Access control policies: some unanswered questions. Comput. Secur. **8**, 43–54 (1989)
69. T.F. Lunt, P.G. Neumann, D.E. Denning, R.R. Schell, M. Heckman, W.R. Shockley, Secure distributed data views security policy and interpretation for DMBS for a Class A1 DBMS. Tech. Rep. RADC-TR-89-313, vol. I, Rome Air Development Center, Griffiss, Air Force Base, NY, December 1989
70. J. McLean, Security models and information flow, in *Proceedings of the IEEE Symposium on Security and Privacy* (IEEE Comput. Soc., Los Alamitos, 1990), pp. 180–189
71. J. Millen, The cascading problem for interconnected networks, in *Fourth Aerospace Computer Security Applications Conference*, 1988, pp. 269–273
72. J. Murray, An exfiltration subversion demonstration. Master's thesis, Naval Postgraduate School, Monterey, CA, June 2003
73. S. Myagmar, A. Lee, W. Yurcik, Threat modeling as a basis for security requirements, in *Proc. Symp. Requirements Engineering for Information Security (SREIS 05)*, 2005
74. P. Myers, Subversion: the neglected aspect of computer security. M.S. thesis, Naval Postgraduate School, Monterey, CA, 1980
75. National Computer Security Center, Trusted network interpretation of the trusted computer system evaluation criteria, NCSC-TG-005, July 1987
76. National Computer Security Center, A guide to understanding object reuse in trusted systems. Tech. Rep. NCSC TG-018, National Computer Security Center, Fort George G. Meade, MD, 1991
77. E.I. Organick, *The Multics System: An Examination of Its Structure* (MIT Press, Cambridge, 1972)
78. L.C. Paulson, *Isabelle: A Generic Theorem Prover*. LNCS, vol. 828 (Springer, Berlin, 1994)
79. V. Paxon, Bro: a system for detecting network intruders in real-time. Comput. Netw. **31**(23–24), 2435–2463 (1999)
80. D. Redell, R. Fabry, Selective Revocation of Capabilities, *International Workshop on Protection in Operating Systems*, IRIA, 1974
81. D. Rogers, A framework for dynamic subversion. Master's thesis, Naval Postgraduate School, Monterey, CA, June 2003
82. A. Roscoe, CSP and determinism in security modelling, in *Proceedings of the IEEE Symposium on Security and Privacy* (IEEE Comput. Soc., Los Alamitos, 1995), pp. 114–127
83. J. Rushby, Design and verification of secure systems. *ACM SIGOPS Operating Systems Review*, vol. 15, December 1981, p. 12
84. J. Rushby, S. Owre, N. Shankar, Subtypes for specifications: predicate subtyping in PVS. IEEE Trans. Softw. Eng. **24**(9), 709–720 (1998)
85. J.H. Saltzer, M.D. Schroeder, The protection of information in computer systems. Proc. IEEE **63**(9), 1278–1308 (1975)
86. R. Sandu, Analysis of acyclic attenuating systems for the SSR protection model, in *Proceedings of the 1985 IEEE Symposium on Security and Privacy*, April 1985, pp. 197–206
87. R.S. Sandhu, The schematic protection model: its definition and analysis for acyclic attenuating schemes. J. ACM **35**, 404–432 (1988)
88. R.R. Schell, P.J. Downey, G.J. Popek, Preliminary notes on the design of secure military computer systems. Tech. Rep. MCI-73-1, Electronic Systems Division, Air Force Systems Command, Hanscom AFB, Bedford, MA, 73
89. R. Schell, T.F. Tao, M. Heckman, Designing the GEMSOS security kernel for security and performance, in *Proceedings 8th DoD/NBS Computer Security Conference*, 1985, pp. 108–

119
90. D.D. Schnackenberg, Development of a multilevel secure local area network, in *Proceedings of the 8th National Computer Security Conference*, October 1985, pp. 97–101
91. M.D. Schroeder, J.H. Saltzer, A hardware architecture for implementing protection rings. Commun. ACM **15**(3), 157–170 (1972)
92. J.S. Shapiro, J.M. Smith, D.J. Farber, EROS: a fast capability system, in *SOSP'99: Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles* (ACM, New York, 1999), pp. 170–185
93. L.J. Shirley, R.R. Schell, Mechanism sufficiency validation by assignment, in *Proceedings 1981 IEEE Symposium on Security and Privacy*, Oakland (IEEE Comput. Soc., Los Alamitos, 1981), pp. 26–32
94. W.R. Shockley, R.R. Schell, TCB subsets for incremental evaluation, in *Proceedings Third AIAA Conference on Computer Security*, December 1987, pp. 131–139
95. A. Silberschatz, P.B. Galvin, G. Gagne, *Operating System Concepts*, 7th edn. (Wiley, New York, 2005)
96. Snort.org, Snort. http://www.snort.org/, last referenced 22 March 2009
97. Specware 4.2 Manual, Kestrel Technology, http://www.specware.org/documentation/4.2/languagemanual/SpecwareLanguageManual.pdf, 3 November 2008
98. J.M. Spivey, *Understanding Z: A Specification Language and Its Formal Semantics* (Cambridge University Press, Cambridge, 1988)
99. D.F. Sterne, On the buzzword "security policy", in *Proceedings of the IEEE Symposium on Research on Security and Privacy*, Oakland, CA (IEEE Comput. Soc., Los Alamitos, 1991), pp. 219–230
100. The Easter Egg Archive, Excel Easter Egg—Excel 97 flight to credits. http://www.eeggs.com/items/718.html, last accessed 19 February 2009
101. K. Thompson, Reflections on trusting trust. Commun. ACM **27**(8), 761–763 (1984)
102. S. Trimberger, Trusted design in FPGAs, in *Proceedings of the 44th Design Automation Conference*, San Diego, CA, June 2007
103. US Department of Commerce and Communications Security Establishment of the Government of Canada, Implementation guidance for FIPS PUB 140-2 and the cryptographic module validation program, initial release: 28 March 2003, last update: 10 March 2009. National Institute of Standards and Technology, Gaithersburg, MD, March 2009
104. US Department of Commerce, Security requirements for cryptographic modules, Federal Information Processing Standards Publication 140-2. National Institute of Standards and Technology, Gaithersburg, MD, May 2001
105. US Department of Commerce, Standards for security categorization of federal information and information systems, Federal Information Processing Standards Publication 199. National Institute of Standards and Technology, Gaithersburg, MD, February 2004
106. US Department of Commerce, Recommended security controls for federal information systems, NIST Special Publication 800-53 Revision 2. National Institute of Standards and Technology, Gaithersburg, MD, December 2007
107. US Department of Commerce, Security requirements for cryptographic modules, Federal Information Processing Standards Publication 140-3 (Draft: 07-13-2007). National Institute of Standards and Technology, Gaithersburg, MD, July 2007
108. US Department of Commerce, Security considerations in the system development life cycle, NIST Special Publication 800-64 Revision 2. National Institute of Standards and Technology, Gaithersburg, MD, October 2008
109. US Department of Commerce, Derived test requirements for FIPS PUB 140-2, Security requirements for cryptographic modules, 24 March 2004, Draft, CMVP program staff (NIST, CSE and CMVP laboratories). National Institute of Standards and Technology, Gaithersburg, MD. http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/fips1402DTR.pdf. Cited 7 April 2009
110. US Department of Defense, Trusted computer systems evaluation criteria (Orange Book) 5200.28-STD. National Computer Security Center, Fort Meade, MD, Dec. 1985
111. US Department of Defense, A guide to understanding trusted distribution in trusted systems,

version 2, NCSC-TG-008. National Computer Security Center, Fort Meade, MD, December 1988
112. US Department of Defense, A guide to understanding trusted recovery in trusted systems, version 1, NCSC-TG-022. National Computer Security Center, Fort Meade, MD, December 1991
113. US Department of Defense, Defense Science Board task force on high performance microchip supply. Office of the Under Secretary of Defense For Acquisition, Technology, and Logistics, Washington, DC, February 2005
114. US Department of Defense, TRUST in integrated circuits, presolicitation notice, solicitation number: BAA07-24. Defense Advanced Research Project Agency, Microsystems Technology Office, Arlington, VA, March 2007. http://www.darpa.mil/mto/solicitations/baa07-24/index.html, cited 27 Mar 2009
115. D. Volpano, C. Irvine, Secure flow typing. Comput. Secur. **16**(2), 137–144 (1997)
116. D.R. Wichers, Conducting an object reuse study, in *Proceedings of the 13th National Computer Security Conference*, October 1990, pp. 738–747
117. M.V. Wilkes, R.M. Needham, The Cambridge model distributed system. ACM SIGOPS Oper. Syst. Rev. **14**(1), 21–29 (1980)
118. E. Witchel, J. Cates, K. Asanovic, Mondrian memory protection, in *Tenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, San Jose, CA, October 2002
119. C. Zymaris, A comparison of the GPL and the Microsoft EULA. 2003. Cybersource. Retrieved 15 September 2008, from http://www.cybersource.com.au/cyber/about/comparing_the_gpl_to_eula.pdf

第3章　硬件安全的难点

摘要：本章主要讨论FPGA产品中的恶意硬件，或者门件（gateware）所带来的问题。同时还讨论了恶意硬件的分类、晶圆代工厂的可信度，以及由植入的恶意硬件发起的攻击。本章还阐述了FPGA中的隐蔽信道问题，并对FPGA器件中一般的隐蔽信道进行正式定义，介绍FPGA器件中隐蔽信道的具体例子。同时讨论探测并消除这些隐蔽信道的方法。

3.1　恶意硬件

第2章中针对高可信软件讨论所获得的相关结果同样适用于FPGA的设计过程。之所以从软件开始讨论，是因为已完成的很多和软件相关的计算机安全工作可供借鉴。由于现代FPGA设计过程在很多方面均类似于软件开发（硬件描述语言和高级程序语言对应，IP复用等），现在已经开始使用门件这个术语来描述载入到FPGA中的电路设计。

恶意硬件的历史可以追溯到冷战时期，当时美国和苏联互相刺探情报并一直延续到了今天。

3.1.1　恶意硬件的分类

攻击者可以把恶意功能植入硬件或者软件之中。恶意硬件和恶意软件有很多相同点，恶意软件的分类方式也适用于恶意硬件。对付恶意硬件是很困难的，一般情况下很难确定计算机程序（或者硬件模块）的可信度，因为根据Rice定理[31]，这种分析等同于挂起问题。规避恶意硬件需要采用安全的设计方式，包括强制的访问控制机制、安全系统的形式验证和配置管理。

后门或者暗门允许未获得授权的用户访问某系统。这些功能可以在系统开发阶段被植入，也可以在系统升级阶段被植入。

破坏开关是另一种破坏性的植入体，攻击者只要操纵破坏开关就能禁用硬件或软件的某些功能[5]。在系统的开发或系统维护过程中可以与暗门一样植入破坏开关，但其破坏功能不是由授权其执行非法访问，而是由于其拒绝执行某些服务（DoS）而造成的。破坏开关可以作为比特流的一部分被植入芯片中，也可以由第三方开发工具，或者由海外晶圆代工厂内的敌方人员植入到芯片中。

已经证实，部分FPGA病毒能够对FPGA器件进行配置，使其发生短路，最终

导致器件的损毁[11]。病毒可以通过软件（带有恶意 FPGA 配置信息的恶意软件）或者硬件复制（比软件更加困难）进行传播。尽管 FPGA 病毒并不常见，但对于敏感性应用领域而言，必须考虑到可能性极小的事件。就像所述的 FPGA 比特流的成功逆向工程一样。尽管非常难以实现，但在特定的环境中必须加以考虑。

硬件木马是硬件安全行业中的一个术语[38]，用来表明在集成电路中存在恶意的更改或者被植入恶意的组件。王（Wang）、特仑尼普（Tehranipoor）和布勒思奎立克（Plusquellic）开发了一个可以对集成电路中的恶意更改进行分类的框架。该框架可用于评估发现恶意更改[38]。王（Wang）等人根据其物理特性、激活方式以及行为特征对硬件木马进行分类。把其中的物理特性进一步划分为以下四种属性：①类型（通过添加还是删除晶体管或门器件来实现）；②规模（添加，删除或者弃用的芯片部件的数量）；③分布（恶意组件在芯片的物理布局中是集中分布还是离散分布的）；④结构。Wang 等人提出的分类学中使用了四个物理属性、一个激活属性和一个行为属性，共计六个属性[38]。硬件木马可以从外部激活或从内部激活，木马行为可以分为三大类：①篡改芯片的功能；②篡改芯片的参数属性（例如，延迟时间）；③将关键信息传输给攻击者。发现硬件木马和减轻它的破坏作用是目前研究的热点。

3.1.2 晶圆代工厂的可信度

在硬件可信度问题中，国防高级项目研究机构（DARPA）启动了一个被称为集成电路可信度（TIC）的计划，通过使用破坏性（例如打磨和扫描）以及非破坏性（例如 X 射线）技术来探测 ASIC 芯片中恶意植入的组件[32]。这个可信度计划还同时考虑 FPGA 安全性以及第三方知识产权（IP）等问题。美国国防部以及国家安全局（NSA）已经启动可信晶圆代工厂计划，用于识别用在敏感性政府系统中芯片的安全性问题，几家晶圆代工厂已经通过了验证[9,23]。

由政府部门确立的 TIC 中还考虑这样的问题，即不可能全由值得信赖的晶圆代工厂加工重要系统中使用的所有芯片。例如，在一架战斗机中，可能需要使用数百个处理器。由于军方必须节省支出，在现实中，使用商用器件是不可避免的。因此在 TRUST 计划中又进一步分为数个小组或者团队，每个小组都由合作的学术机构和工厂组成[32]。其中由政府资助的研究小组包括红色小组，测试样品生成小组，以及标准制定小组。其中测试样品生产小组由南加州大学信息科学研究院（USC-ISI）牵头，使用 MOSIS 在商业代工厂中加工测试芯片。红色小组由麻省理工学院林肯实验室牵头，主要目的在于识别各种恶意电路，以及将这些恶意电路植入测试芯片中的开发方法。其中有三个研究小组负责侦查电路中是否有恶意植入的组件：Xradia 小组主要负责非破坏性 X 射线侦查技术；Luna 小组主要负责 FPGA 器件的防篡改功能；Raytheon 小组主要负责硬件以及逻辑电路的测试[5]。约翰霍普金斯大

学应用物理实验室的度量规则小组，主要负责制定测试结果的度量规则（即怎样才算成功地完成芯片的测试）。在计划的实施过程中，执行团队必须满足严格的侦查率以及误报率指标。除了在军用系统的集成电路中植入恶意组件问题之外，信任（TURST）计划还考虑了使用第三方 IP 的 ASIC 和 FPGA 设计流程中可能存在的问题[32]，使用第三方 IP 可能会导致严重的问题。

Trimberger 讨论了晶圆代工厂的可信度问题，因为此问题和 FPGA 相关[36]。由于代工厂并不知道哪片芯片需要交付到哪个具体的客户手中，因此很难执行目标明确的攻击。另外，由于 FPGA 是在加工完成后才进行编程的，代工厂也并不知道在相关器件中会载入哪种设计。FPGA 器件出色的可编程能力让代工厂无法获得关于其设计的信息。例如，CPU 的可编程能力就低于 FPGA，因此攻击者知道 CPU 执行程序的方式以及 CPU 的结构，这就使得其更容易成为攻击的目标。攻击者可以将 CPU 中的特定部件确定为攻击目标，只需要约 1000 个门器件就可以让攻击者完全控制整个系统[17]。另一方面，FPGA 是可编程逻辑器件的阵列，晶圆代工厂内的恶意攻击者无法预测在 FPGA 上会加载何种应用程序，这也会提高攻击的难度。尽管如此，在 FPGA 中包括一些专用于执行具体功能的部件，这些部件就容易成为恶意攻击的目标。例如，用于加载比特流或者用于存储密钥的 CPU 硬核，SRAM 硬核和 BRAM 模块，以及标准的输入/输出功能模块。

对于防御方而言，可重构性可以让核和数据在芯片中移动，从而能够进一步增加攻击者的难度。防御方能够使用冗余来进一步扩大优势，迫使攻击者必须以完全相同的方式修改所有的冗余版本。显然，这些理念有点类似于用模糊性换取安全性，系统的安全性不应该只靠蓝图保密。尽管如此，若将保密工作与更加正式、有效的方法结合，便能够增加攻击系统的成本。

设计提示：通过模糊性获得安全性。不要让系统的安全性完全依赖于系统蓝图的保密，应该考虑敌人也许能够窃取蓝图或通过逆向工程破译系统的构造。密码系统必须具有密钥，但所用的加密算法是公开的，因此，全世界的密码从业人员都可以仔细地研究如何破译密钥，揭开 FPGA 安全机制蓝图的秘密。如果系统存有商业秘密，且其安全性十分依赖保密机制是否严密，则应挑选能为你严格保密的生产商。

3.1.3 物理攻击

在恶意组件被有意植入电路中后，就为在后续发动针对该电路的攻击（无意的设计缺陷也会导致系统中存在漏洞）提供了可能性。在物理攻击情况下，攻击者能够实现针对该器件的物理控制。根据攻击对目标系统的物理侵入程度，攻击方

式可以分为如下三类：①非侵入性；②半侵入性；③侵入性。例如，如果攻击者能够对智能卡终端实现物理控制，则其完全有可能对此智能卡终端发动非侵入性的旁路攻击。由于智能卡完全依赖于终端提供的电源，因此相关攻击可以采取简单的功率分析、功率差分析，或者针对加密电路进行错误注入以便获得密钥。功率分析攻击也可以在 FPGA 上进行[34]。需要拆除封装，但不会对芯片造成任何物理更改或者损坏的旁路攻击方式被称为半侵入攻击。封装拆除之后，攻击者可以更容易地分析由芯片自动发射的辐射信号。侵入式攻击的一个例子就是逐层研磨和扫描攻击。在进行这种攻击时，需要将集成电路的保护层完全拆除，并采用电子显微镜进行扫描[7]。可以采用化学物质、激光或者聚焦离子束来拆除保护层。侵入式攻击的另一个例子是使用化学溶剂来去除智能卡的封装，之后采用探针来探测智能卡内总线的数据流量。非常专业的物理攻击需要使用聚焦离子束在智能卡周围的封装材料上钻孔，之后将采用金属分流器来探测智能卡的处理器，在这个过程中不会触动处理器周围的防篡改机制。

设计提示：防篡改机制、比特流加密，以及确定攻击者。在掌握足够资源的情况下，攻击者可以战胜相关的防篡改机制。业余人员就知道如何绕开智能卡的防篡改机制，以便免费收看卫星电视节目[6]。在你的风险评估过程中，应该考虑到针对比特流探测机制的功率差分析。在选择 FPGA 平台时，对于由不同供货商提供的比特流加密机制进行彻底的安全评估是非常有用的。

和简单的板级攻击对比（探测电路板上的金属引脚），逐层研磨和扫描攻击是器件级别或者芯片级别攻击的最典型例子。安全设计人员通常在部分或者全部芯片上（例如处理器或者缓冲区）设置逻辑安全区或者安全边界，以便进行分析和记录。例如，在硬件设计方式中，可能会在数据离开芯片的安全区域之前对其进行加密处理。在其重新进入安全区域之后再对其进行解密，同时在数据重新进入安全区域之后，还将对数据的完整性进行检查[18,22,24,25,39]。

设计提示：安全区域。必须警惕这样一种想法，即在部件周围设置了安全区或者安全边界之后，则安全区域内的所有东西都是安全的。必须认识到在掌握足够资源的情况下，特定的攻击者能够攻破任何上述的“马其诺防线”。

规避和应对物理攻击是非常困难的，在建立威胁模型时，非常重要的一点是必须考虑攻击者能够掌握的资源情况。尽管部分物理攻击能够由业余者在五金店内购

买的设备实现，但相当一部分物理攻击需要非常专业的知识和设备。例如，聚焦离子束工作站在半导体行业中的使用是合法的，若攻击者想通过购买这样的设备用于执行特定类型的物理攻击，需要花费数百万美元的成本。采用纳米尺寸的加工方式也能够增加攻击者的难度。很多针对 CPU 和 ASIC 开发的技术也同样适用于 FPGA。例如针对 IBM 4758 开发的防篡改机制，在此机制中，CPU 的周围环绕着对篡改非常敏感的线网和环氧封装材料[33]。

3.2 隐蔽信道定义

执行安全策略时，最先应考虑的是对诸如进程和内核等动态实体实施强制隔离，并确保相关动态实体之间的通信只能在被允许或者被控制的前提下，有序地进行。把各种共享物理资源做局部虚拟化处理，可以实现资源的隔离。通过这种方式可为每个进程或内核提供一个不会受到其他实体干扰的独特的虚拟资源。

3.2.1 进程抽象

对通用型处理器而言，进程抽象能够通过将各种进程元素，例如寄存器、线程、程序计数器、代码段和程序数据段等整合为一个能够推理出计算机行为的实体，用以简化软件的开发过程。进程中可能包括多个线程和环，其中每个都具有独特的安全特征，因此，主体抽象（即进程/环对）经常被用于更加精确地推断与安全相关的动态实体。同样，多核处理器中一个专用的单一用途内核可以被视为一个主体。

3.2.2 等价类

安全系统设计的另一种简化方法是将相似的实体归入同一个域（或等价类）中，以减少需要跟踪事物的数量。多域安全策略通过为每个主体绑定一个敏感标签，把系统的每个主体和客体分开，将它们归入不同的等价类。虚拟机和多核处理器也可以进行相关配置，将计算机资源划分为不同的等价类，这样具有相似安全属性的所有进程都可以被分配到相同的虚拟机（VM）或者核中，或者根据策略要求，把多个核分配给同一个等价类。这可使访问控制机制的设计人员能够将注意力集中在那些跨越等价类边界的主体行为上。

设计提示：等价类。等价类抽象能够让你的设计更加安全，更加有效。把相似的主体和客体归类、分组，放入多个域中，能够降低策略的复杂度，降低执行机制的开销，同时简化安全分析过程。

3.2.3　形式定义

隐蔽信道是指不同主体间信息传递的一种手段，但这种信息传递的目的不是信息交流，并且系统安全策略[15]也不允许进行这种信息传递。由于资源的虚拟化还不够彻底，不同主体间很容易发生资源争夺（例如，磁盘是否已被写满，或者处理器是否已被占用），这是显而易见的。这种虚拟资源的不彻底性常引发资源争夺，在争夺冲突点的周围很可能产生一个隐蔽信道。根据隐蔽信道接收器检测到的冲突方式可以区分出该隐蔽信道究竟是（隐蔽）存储信道，还是（隐蔽）时序信道（前者接收器检测到的是各种不同的系统调用状态信息，而后者接收器检测到的是相对事件的时序变化）。

3.2.4　同步

通过隐蔽信道的数据传送需要发送端（即强制保密策略中保密度较高的端点）和接收端（即保密度较低的端点）操作的精确同步，以便重复冲突事件。不需要保密度较高端同步的隐蔽信道（换而言之，不需要保密度较高的发送端主体中的恶意功能或者特洛伊木马的配合，就可以通过保密度较低的接收端窃听到高端的行为）被称为侧信道。在这种场合，由于不能提供未获授权数据传输的同步操作，因此侧信道通常特指位宽固定的数据，例如加密密钥。

3.2.5　共享资源

为了增强性能，现代的处理器都试图通过在微体系结构层面，使用在线程、内核，以及处理器之间共享的方式，实现资源的完全利用（例如，指令缓存、数据缓存、浮点算法单元、以及分支预测单元）[1-4,14]。因此需要操作系统对这些资源进行虚拟化处理，以防止属于不同等价类的主体之间互相干扰。例如，即使 L1 和 L2 缓存从逻辑上处于隔离状态，在虚拟化不彻底的情况下，L3 缓存会提供一个两者之间的干扰点。在缓存的侧信道中，一个主体使用特定的缓存行来增加下一个主体使用其的响应时间，即使两个主体在不同的内存地址上使用此缓存行，也可以导致用于加密的数据暴露，最终导致密钥暴露。此外，如果浮点单元的使用存在互斥性，则在有一个主体使用时，其他主体就会表现为出现延迟。

3.2.6　要求

一般情况下，隐蔽信道（在文献中有很多不同的称呼）基于如下五个因素：

1）受害者和接收端处于不同的等价类中（如上所述）。

2）受害者（发送端）以及攻击者（接收端）之间共享资源，例如共享缓存行集。

3）具备对发送端和接收端进行初始化并同步其行为的手段（通常攻击者能够根据其意愿随时发动攻击行为，如同在企业内部的端到端VPN）。对于侧信道而言，发送端通常完全意识不到攻击行为，因为这种攻击行为不需要同步。

4）具备探测是否有其他主体正在使用或者使用过相关资源的手段（攻击者通过缓存行读取内存时在响应时间上的差别）。

5）对通过侧信道获得的信息进行解释可能需要了解密钥和受害者加密功能中内存结构使用之间的相关关系，例如密码置换盒（S-box）。

3.2.7 旁路

除了隐蔽信道和侧信道之外，还存在直接信道，这种信道也被称为公开信道或者旁路。简而言之，直接信道就是没有（或者未采用）任何安全机制来防止系统中两个实体间进行通信[21]的信道。直接信道通常具有较高的带宽。例如，在没有内存保护机制的系统中，两个内核能够通过写入和读取外部存储器某个特定的共享存储器区进行通信。直接信道的另一个例子是系统中具有内存保护机制，但两个内核可以绕开内存保护机制，通过外部存储器进行直接通信。再举一个直接信道的例子，在总线上没有仲裁机制防止非法通信的情况下，带有多个内核的系统允许通过共享的总线进行通信。

3.3 制约隐蔽信道和侧信道攻击的现有方法

在共享资源的微结构设计过程中，若操作系统设计师没有按照工业标准结构（ISA）的基元（即基本模块）来保证共享资源的安全性（例如，Intel iAPX86处理器中的任务及区段管理功能就是这种基元的例子），则受命设计硬件安全管理机制的操作系统设计师将面临十分困难的境地。以前的软件方法不是忽略此问题，就是采用过分严格的策略，从而严重影响系统的性能。

3.3.1 共享资源矩阵法

Kemmerer[15,16,21]提出了一种识别计算机系统中隐蔽存储信道以及隐蔽时序信道的共享资源矩阵法。在矩阵中枚举了所有会被某个主体访问或者修改的共享资源，之后再对这些共享资源进行检查，确定其是否会被某个主体用于向另一个主体以隐蔽的方式传递信息。共享资源矩阵中的行代表所有的共享资源和这些共享资源对主体可见的属性。每种操作方式为矩阵的一列。通过分析矩阵即可以确定能够访问或者修改相关属性的操作方式。当接收端和发送端都能访问共享主题的同一个属性时，矩阵会将其显示出来，并指定发送端可以更改这个属性，接收端只能访问这个属性。正如前文相关内容所述，对于隐蔽信道而言，还需要一个初始化接收端和

发送端，并拟定接收端和发送端访问顺序的机制。

3.3.2 缓存干扰

单核处理器的缓存干扰通常由对不同安全域之间缓存（例如，进程上下文切换过程中的缓存）的归一化（即擦除所有缓存行）来处理。从片外存储器转入缓存需要较长的时间，因此现在已经开发出了很多技术来避免不必要的重复操作[12]。尽管如此，对于支持并发执行的系统［片上多处理器（CMP），带有同时多线程（SMT）的单核计算机，以及具有缓存一致性机制的对称多处理器（SMP）系统］而言，这种方法相对较为低效，这主要是由于访问这些缓存时，需要采用比进程切换更加精细的交错访问方式。在这些系统中，微体系结构干扰是个巨大的挑战，因为其有效的解决方案已经超过了常规操作系统虚拟化技术的能力范围。最近的研究得出了多种结果，例如，如果硬件支持虚拟缓存，可以根据策略等价类对缓存进行物理分区或者逻辑分区[30,37]。在物理分区的情况下，这就需要对处理器进行修改，同时需要降低物理分区以及逻辑分区的有效缓存尺寸。尽管存在多种类型的缓存禁用技术，例如将缓存关闭、将其针对特定的核或者进程关闭等[28]，但所有这些技术都需要以消耗响应时间为代价。降低缓存信道的带宽[29,30,35]有可能使设计留有未来被攻击者利用的漏洞，因为这种方法并没有从根本上消除隐蔽信道的干扰。应用层级别针对具体加密算法的安全防范措施也存在相同的问题[1]。

3.3.3 FPGA 掩码的保护方法

尽管以前已经针对时序、功率以及电磁侧信道开展了大量的工作[10,19,20]，其中部分结果也适用于 FPGA，但 ASIC 用于抵抗侧信道攻击的技术并不一定适用于 FPGA，因为 FPGA 和 ASIC 的逻辑门和电路在实现方式上存在许多差别。Yu 和 Schaumont 开发了一种能够抵抗侧信道攻击的 FPGA 设计方式[40]。其技术包括构建一个补充性的对称电路来掩蔽功率消耗情况，防止其被复制。Chen 和 Schaumont 也提出了一种硬件掩蔽方法，这种方法结合了算法掩蔽以及多数位掩蔽方法，使得攻击者更难在电路级别上预估随机的内部安全掩码[8]。

3.4 FPGA 隐蔽信道攻击的探测及应对

FPGA 器件的功能目前已十分强大，本科生就能使用商用 FPGA 开发板和设计工具，来构建嵌入式片上系统（SoC）。一个设计通常需要用到很多个 IP 核，每种 IP 核有特定的功能。正如软件的情况一样，硬件也存在安全问题，有漏洞的 IP 核很可能泄露信息。

3.4.1　设计流程

在设计阶段中，通过所用设计工具或 IP 核中故意设置的漏洞（类似于在编译器或共享库中故意设置的漏洞），可以在系统中植入隐蔽信道、侧信道和直接信道。与软件领域的情况一样，FPGA 的设计也涉及很多个阶段，每个阶段都十分复杂。有很多个设计流程，不同的设计流程涉及各种不同工具的组合，一种工具只是大型设计流程中的一个阶段。与软件领域的情况完全一样，代码复用对于设计成本的管理至关重要，在很多 FPGA 设计流程中经常复用 IP 核。由于当前设计项目的规模以及复杂度都极高，因此不可能对系统中所包含的每个部件都从头开始设计。

3.4.2　空间隔离

针对 FPGA 隐蔽信道的攻击问题，本书提出的解决方案利用了 FPGA 的可重构性。不论需要何种安全机制，设计师都可以用 FPGA 的可重构硬件实现这种安全机制。这些解决方案同时还利用了 FPGA 的空间特征，这个特征让设计师可以把 IP 核布置在 FPGA 中的特定区域内。本书第 6 章中提出了一种在布局阶段设计时，将不同 IP 核进行空间隔离，以防止不同 IP 核互相干扰的技术。这种技术称为壕沟与吊桥技术，壕沟用于隔离 IP 核，而吊桥则用于以明确受控的方式提供资源共享。本书第 6 章中还提供了多核系统中隐蔽时序信道的具体实例，同时提供了针对此问题的解决方案，在此系统中不同内核间共享总线并利用了 TDMA 总线仲裁机制实现通信。

3.4.3　存储保护

在本书第 5 章中，将讨论用于防止不同核互相干扰的存储保护技术。具体地说来，就是通过在可重构硬件中嵌入参考认证机制，规定不同核共享存储器的安全策略。通过使用参考监视器能够拒绝所有 IP 核提出的非法存储访问要求，能够防止片外部存储器被用作不同 IP 核之间通信的手段。同时采用精确定义的语言来描述存储访问策略，并利用设计流程将相关策略直接编译为可重构的参考监视器的电路，来执行相关的策略。

3.5　作为隐蔽存储信道的策略状态

正如本书第 5 章将详细叙述的那样，参考监视器能够根据其执行的具体安全策略要求，决定授权或者拒绝特定的存储器访问请求。共有两种类型的策略：①无状态策略，即只有一种状态的策略；②有状态策略，即有两种及两种以上状态，不同状态之间会发生转换的策略。对于特定的有状态策略而言，参考监视器的内部状态会被隐蔽存储信道用作攻击共享资源。高级别的发送端 IP 核，可以通过以接收端

IP 核能够观察到的方式，对参考监视器的内部状态进行修改，从而实现向低级别接收端 IP 核发送信息。引用和修改参考监视器内部状态的能力，可能会被用于非法通信[13]。

3.5.1 状态策略

状态策略可以表述为有向图，图中包含有多个状态（即节点）之间的状态转移（定向边）。有些状态策略图中画有循环圈，若满足一定条件，则每个循环圈代表一个潜在的隐蔽信道。在这种场合，最保守的做法是修改状态策略，以便消除循环圈。然而，若无法修改状态策略，则可以对循环圈的使用情况进行监视，以主动应对隐蔽信道。若在特定时间内，该隐蔽信道的使用率超过了某个特定的阈值，则应该采取相应的措施。用计数器检测使用率，每完成一个循环，计数器的读数就增加一，表明有 1bit 的信息流动。可采取的几种纠正措施将在下文中详细讨论。

3.5.2 隐蔽信道机制

每条隐蔽信道中都包括一个发送器和一个接收器，在多层交换（MLS）系统中，发送器的保密标签级别高于接收器。在支持多层交换（MLS）策略的 FPGA 系统中，不同的 IP 核位于不同的安全等级，发送器和接收器都是 IP 核。发送器通过发送访问请求使得策略状态发生跳转，进而改变参考监视器的内部状态。如果只允许接收器在一个状态访问监视器一次，而不允许在其他状态时访问监视器，则接收器便能够观察到此变化。也就是说，这两个状态对于接收方来说，一个允许，另一个不允许。由于传输字母表中包括两个字符（即图 3-1 中的白色和黑色两个状态），这就允许发送器发送 1bit 信息流至接收器。根据这种干扰机制构建的信息流必须在

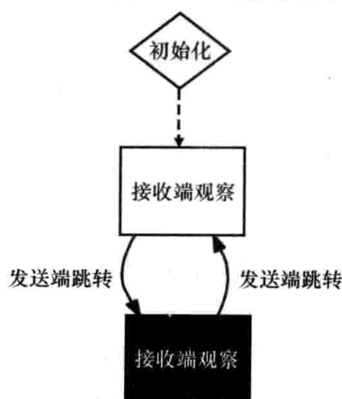


图 3-1　状态策略可以通过定向图的方式表述，其中包含不同状态（节点）之间的跳转（定向边）。此图形中包括有一个循环，说明存在潜在的隐蔽信道。发送端可以在两个状态之间自由切换，接收端可以观察到两个状态之间的差别

状态图中含有涉及两个或两个以上状态的循环。每完成一次循环，就发生1bit信息的流动，然后状态重置以便进行下一个循环。在理想情况下，循环只涉及两个状态，而该循环过程中的两次状态转移都是由发送器导致的，如图3-1所示。实际上，即使一个很大的循环并且包含了很多状态，发送器在循环过程中也只涉及一次状态转移，发送器只有等待足够长的时间，才能执行下一次状态转移，并重新开始循环。但接收器可能会触发循环中的其他跳转。即使在较大的循环中，对于接收器而言，必须区别的状态也只有两个。

3.5.3　编码方案

对于想要发送的数据，可以采用多种编码方案。有一种方案是让其中的某个状态在固定个数的时钟周期内保持不变，利用状态变化之间的时间长度作为信号代码。另一种方式是将一个完整的循环作为一个比特位，类似于Morse编码脉冲。这样，根据可能编码的数据个数以及每个字符传输超时的概率，便可以计算出隐蔽信道的带宽[26,27]。请注意，这一点非常重要，在系统运行时，并非所有的隐蔽信道都可以被敌方利用，必须做进一步的分析，以消除误判。

3.5.4　隐蔽存储信道探测

有一种简单的静态技术，可用于分析检测隐蔽存储信道是否存在的策略，这种技术首先采用拓扑分类来确定状态图中是否存在任何循环。可能导致状态循环转移的任何IP核都可能成为发送器，同时，能够观察到两个节点之间状态差异的任何IP核都有可能成为接收器。采用简单静态技术设计的探测器已被证明可用于分析检测隐蔽存储信道是否存在的多种策略。该探测算法的伪代码如下：

```
Procedure DetectChannels(Graph G)
{
 Array of Lists Senders
 Array of Lists Receivers
 If(Topological_Sort(G) == False)
  Output"Graph G Contains No Cycles."
  Return
 C = Recursively_Trace_Graph_to_Find_Cycles(G)
 For (All Cycles C)
  For (All Edges E in C)
   M = Module that causes transition E
   Add M to Senders[C]
  For (All Vertices V in C)
```

```
        For ( All Vertices V'in C) if V'! = V then
          For ( All Rows r)
            For ( All Columns c)
              If ( Matrix( V) [ r] [ C] ! = Matrix( V') [ r] [ c])
                Add c to Receivers[ C]
    Output " Possible Covert Channels:"
    For ( All Cycles C Found)
     Output Cross _ Product( Senders[ C] , Receivers[ C] )
    Return
}
```

Matrix(V) 为节点 V 的访问矩阵。Senders[C] 是循环 C 中可能引起转移到状态 E 的模块清单。Receivers[C] 是能够观察到循环 C 中任何两个节点 V 以及 V′的访问矩阵之间差别的模块清单。

3.5.5 减轻隐蔽信道可能造成的危险

在安全策略层次上，一旦发现可能存在隐蔽信道，从降低风险的角度看，最好的办法就是修改安全策略，以消除有问题的循环。如果上述方式不可行（例如，这么做会导致关键服务被禁用），则系统设计师可以在一个时间滑动的窗口中，用计数器检测并跟踪相关循环的发生次数，从而限制信道的带宽。若测量窗口中的带宽超过了阈值，则系统将转到另一个没有问题的安全循环的策略中去。若有必要，系统还可以在等待一定时间后恢复以前的安全策略。另一种办法是增加状态转移的延迟时间，以便缩小信道的带宽。

参 考 文 献

1. O. Aciíçmez, Yet another microarchitectural attack: exploiting I-cache, in *Proceedings of the First Computer Security Architecture Workshop (CSAW)*, Fairfax, VA, November 2007
2. O. Aciíçmez, S. Gueron, J.P. Seifert, New branch prediction vulnerabilities in OpenSSL and necessary software countermeasures. IACR Cryptology ePrint Archive, Report 039, 2007
3. O. Aciíçmez, J.P. Seifert, Cheap hardware parallelism implies cheap security, in *Proceedings of the Fourth Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Vienna, Austria, September 2007
4. O. Aciíçmez, J.P. Seifert, C.K. Koc, Micro-architectural cryptanalysis. IEEE Secur. Priv. **5**(4), 62–64 (2007)
5. S. Adee, The hunt for the kill switch. IEEE Spectrum **45**(5), 35–39 (2008)
6. R. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems* (Wiley, New York, 2001)
7. R. Anderson, M. Kuhn, Tamper resistance: a cautionary note, in *Proceedings of the Second USENIX Workshop on Electronic Commerce*, Oakland, CA, November 1996
8. Z. Chen, P. Schaumont, Slicing up a perfect hardware masking scheme, in *Proceedings of the 2008 IEEE International Workshop on Hardware-Oriented Security and Trust (HOST-2008)*, Anaheim, CA, June 2008

9. Defense Science Board, High performance microchip supply. *White Paper*, February 2005
10. K. Gandolfi, C. Mourtel, F. Olivier, Electromagnetic analysis: concrete results, in *Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, Paris, France, May 2001
11. I. Hadzic, S. Udani, J. Smith, FPGA viruses, in *Proceedings of the Ninth International Workshop on Field-Programmable Logic and Applications (FPL'99)*, Glasgow, UK, August 1999
12. W.M. Hu, Lattice scheduling and covert channels, in *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, Oakland, CA, May 1992
13. T. Huffmire, T. Sherwood, R. Kastner, T. Levin, Enforcing memory policy specifications in reconfigurable hardware. Comput. Secur. **27**(5–6), 197–215 (2008)
14. J. Kelsey, B. Schneier, C. Hall, D. Wagner, Side channel cryptanalysis of product ciphers. J. Comput. Secur. **8**(2–3), 141–158 (2000)
15. R.A. Kemmerer, Shared resource matrix methodology: an approach to identifying storage and timing channels, in *ACM Transactions on Computer Systems*, 1983
16. R.A. Kemmerer, A practical approach to identifying storage and timing channels: twenty years later, in *Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC)*, Las Vegas, Nevada, USA, December 2002
17. S.T. King, J. Tucek, A. Cozzie, C. Grier, W. Jiang, Y. Zhou, Designing and implementing malicious hardware, in *Proceedings of the First Usenix Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, San Francisco, CA, April 2008
18. D. Kirovski, M. Drinic, M. Potkonjak, Enabling trusted software integrity, in *Tenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, San Jose, CA, October 2002
19. P. Kocher, Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems, in *Proceedings of the 16th Annual International Cryptology Conference (CRYPTO)*, Santa Barbara, CA, August 1996
20. P. Kocher, J. Jaffe, B. Jun, Differential power analysis, in *Proceedings of the 19th Annual International Cryptology Conference (CRYPTO)*, Santa Barbara, CA, August 1999
21. B.W. Lampson, A note on the confinement problem. Commun. ACM **16**(10), 613–615 (1973)
22. D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, M. Horowitz, Architectural support for copy and tamper resistant software, in *Eighth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, San Jose, CA, October 2000
23. J.I. Lieberman, National security aspects of the global migration of the US semiconductor industry. *White Paper*, June 2003
24. J. Lotspiech, S. Nusser, F. Pestoni, Broadcast encryption's bright future. IEEE Comput. **35**(8), 57–63 (2002)
25. J.P. McGregor, R.P. Lee, Protecting cryptographic keys and computations via virtual secure coprocessing, in *Workshop on Architectural Support for Security and Antivirus (WASSA) Held in Conjunction with the Eleventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XI)*, Boston, MA, October 2004
26. J.K. Millen, Covert channel capacity, in *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, April 1987
27. J.K. Millen, Finite-state noiseless covert channels, in *Proceedings of the Computer Security Foundations Workshop II*, Franconia, NH, USA, June 1989
28. D.A. Osvik, A. Shamir, E. Tromer, Cache attacks and countermeasures: the case of AES (extended version). Technical Report, Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel, October 2005
29. D. Page, Theoretical use of cache memory as a cryptanalytic side-channel. Technical Report CSTR-02-003, Department of Computer Science, University of Bristol, June 2002
30. D. Page, Partitioned cache architecture as a side channel defense mechanism. *Cryptology ePrint Archive*, Report 2005/280, 2005
31. H.G. Rice, Classes of recursively enumerable sets and their decision problems. Trans. Am. Math. Soc. **74**, 358–366 (1953)
32. B. Sharkey, TRUST in integrated circuits program: briefing to industry, 26 March 2007.

http://www.darpa.mil/MTO/solicitations/baa07-24/Industry_Day_Brief_Final.pdf
33. S.W. Smith, S.H. Weingart, Building a high-performance, programmable secure coprocessor. Comput. Netw. Int. J. Comput. Telecommun. Netw. (Spec. Issue Comput. Netw. Secur.) **31**(9), 831–860 (1999)
34. F. Standaert, L. Oldenzeel, D. Samyde, J. Quisquater, Power analysis of FPGAs: how practical is the attack? Field-Program. Log. Appl. **2778**(2003), 701–711 (2003)
35. N. Topham, A. Gonzalez, Randomized cache placement for eliminating conflicts. IEEETC: IEEE Trans. Comput. **48**, 185–192 (1999)
36. S. Trimberger, Trusted design in FPGAs, in *Proceedings of the 44th Design Automation Conference*, San Diego, CA, USA
37. Z. Wang, R. Lee, New cache designs for thwarting cache-based side channel attacks, in *Proceedings of the 34th International Symposium on Computer Architecture (ISCA)*, San Diego, CA, June 2007
38. X. Wang, M. Tehranipoor, J. Plusquellic, Detecting malicious inclusions in secure hardware: challenges and solutions, in *IEEE Workshop on Hardware Oriented Security and Trust (HOST)*, Anaheim, CA, June 2008
39. J. Yang, Y. Zhang, L. Gao, Fast secure processor for inhibiting software piracy and tampering, in *Proceedings of the Thirty-Sixth International Symposium on Microarchitecture (MICRO-36)*, San Diego, CA, December 2003
40. P. Yu, P. Schaumont, Secure FPGA circuits using controlled placement and routing, in *Proceedings of the 2007 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'07)*, Salzburg, Austria, October 2007

第4章　FPGA更新及可编程性

摘要：在本章中，将对和FPGA的可编程性相关的安全问题进行讨论。FPGA能够在运行时更改其部分或者全部配置。在本章中，还将叙述如何防止攻击者利用此特点发动攻击。

4.1　概述

与ASIC不同，SRAM型FPGA在其加工完毕后，也能够改变其逻辑配置方式。用于定义逻辑功能的比特流存储在非易失性的片外存储器中，在FPGA上电后加载到FPGA中。这种设计方式的优点在于，在逻辑功能设计中发现缺陷的情况下，可以采用新的比特流更换存在缺陷的比特流，且成本远低于重新制造芯片。另外，比特流可以在FPGA制造完毕之后，通过安全的设施加载到FPGA中，避免了将敏感性IP（后文简称为IP）提交给不能完全信任的晶圆代工厂可能带来的风险。

4.2　比特流加密和认证

将专有的比特流存储在非易失性的片外存储器中可能引起安全问题。FPGA业界投入了很大的努力来开发比特流加密机制，以防止设计被从非易失性存储器中提取出来。对称加密算法用于对存储在非易失性存储器中的比特流进行加密。这种方式能够防止当比特流从非易失性存储器载入到FPGA的过程中，从电路板级对比特流进行探测攻击。解密过程在FPGA内部完成。所以从理论上讲，窃取FPGA的设计需要昂贵的、侵入性的打磨和扫描攻击。这种攻击方式需要利用物理方法来对相关芯片做逆向工程[1]。

Austin[2]首先在一份专利中提出了比特流的加密方法，采用密码和密钥对比特流进行加密处理，并存储在非易失性存储器中。另外，也可以采用嵌入有隐藏签名的数字水印来防止设计失窃。一种FPGA使用的水印方案在物理级别上对比特流未使用的部分进行操纵[8]。另一种保护比特流的方法是在通过安全设施对SRAM FPGA进行编程之后，让其一直处于上电状态。中断电源将会擦除用于加密比特流的密钥。尽管这种方法不需要对比特流进行加密处理，但连续供电的成本较为昂贵，同时无法对FPGA进行重新编程。反熔丝型FPGA具有非易失性，这种器件不需要连续供电，但也不能进行重新编程。Flash型FPGA不需要连续供电，也可以进行

重新编程。

除了加密之外，还需要设置认证机制，以便对比特流的开发人员和向现场器件上传比特流的人员进行身份确认。认证还可以用于防止 IP 在未获得相关授权的 FPGA 上运行。物理不可克隆技术（PUF）[13] 是用于知识产权和硬件双向认证的一种技术[11]。PUF 可以利用不同芯片加工过程中产生的微小差别，为每片芯片分配一个独特的密钥。第三方可以使用 PUF 技术识别相关 IP 核的作者，并设置一个用于认证硬件以及软件的协议。

4.2.1 密钥管理

在实践过程中，密钥管理是一个非常严重的问题。用于解密比特流的密钥总是需要保存在一个地方。如果在加工过程中通过硬线连接的方式将密钥植入到硬件中，则会出现下列两种可能：每片离开生产线的芯片都有相同的密钥，或者每个器件都有独特的密钥。如果所有的芯片都有相同的密钥，则密钥的管理将会非常复杂，因为盗用一片芯片的密钥就能够破解所有的芯片。如果每个器件都有不同的硬线连接密钥，则存储相关器件密钥的数据库将成为对于攻击者非常具有吸引力的目标。在设计过程中面临下列的选择：

1）是不是每个器件都有独特的密钥？

2）密钥是否能够更改？

3）采用何种存储技术来存储密钥（电可擦除可编程只读存储器，由电池供电的静态随机访问存储器等）？

4）在固定的密钥被盗用之后，是否还有可供利用的备用密钥，还是相关的器件必须被放弃使用？

5）密钥是对称的还是非对称的？采用了哪种加密方法？

6）密钥中包括的信息量如何？

7）是否具有相关的密钥管理基础设施？

8）比特流加密机制是以何种方式植入到器件中的？在器件中处于哪个位置？

9）解密模块是否存在发起功率分析或者时序分析攻击的可能性？

部分器件具有在加工完成后可以设置一次的独特密钥。另一种选择是使用物理不可克隆技术（PUF）。尽管如此，使用 PUF 很难生成具有足够长度的可靠密钥。

Actel 60RS 系列 SRAM FPGA 使用了所有器件共享一个相同密钥的方式[9]。在器件加工过程中将一个固定的密钥植入到 FPGA 中。尽管这密钥能够保护器件不会被逆向工程，但其却不能防止器件被克隆。另外，一旦密钥被破解，所有设计都将受到影响，同时密钥信息被嵌入到 CAD 工具中。Xilinx Virtex Ⅱ系列器件将密钥存储在连续供电的易失性存储器中，但不同的器件都有独特的密钥。由于用户存储密钥的寄存器耗能非常少，一块电池就能够支持数年的时间[15]。Virtex 系列禁用了信

息返回功能，能够防止对密钥进行读出或者写入操作，同时防止对比特流的完整性进行检查。

设计提示：密钥管理。采用加密技术并不能保证系统的绝对安全。密钥中必须具有足够的信息量，同时必须对其进行合理的管理。在进行安全分析的过程中，必须考虑到密钥管理基础设施的相关情况，考虑其是否为基于数字证书的公钥基础设施（PKI），或者其他更基础的类型，例如硬线连接对称密钥。加密协议中或者 PKI 实施过程中存在的缺陷可能会被攻击者利用。在分析过程中还需要考虑具体的 FPGA 版本如何实施比特流加密，因为实施过程中存在的缺陷也可能会被攻击者利用。应该采用何种加密技术？解密机制位于哪里？每个器件具有相同密钥、器件组共享密钥还是每个设备独享密钥？密钥是通过硬线连接植入，还是存储在电池支持的易失性存储器中，还是存储在非易失性存储器中，还是利用了 PUF 技术？密钥中包括的信息量是否足够？对解密模块进行时序攻击或功率分析攻击的难易程度如何？是否存在硬线连接的解密模块，或者可以实施可重构解密逻辑？

4.2.2 战胜比特流加密

另一个需要关注的问题是 FPGA 比特流加密机制上的功率分析攻击。这种攻击方式能够分析加密电路的功率消耗情况，进而窃取密钥。这种攻击可以通过增添能够掩蔽加密电路功率消耗的补充电路来加以应对。另一种窃取比特流加密机制的方法是社交工程攻击，也就是通过贿赂 FPGA 公司的工作人员获得密钥或者加密机制的详细信息。保护高价值数据免受攻击者盗取，需要考虑比特流加密机制的强度。这些加密机制的强度可能足够满足商业应用要求，但未必满足处理高价值数据的系统的要求。对于这些应用领域，可能采用反熔丝型 FPGA 比采用 SRAM 型 FPGA 更好，因为相关的逻辑功能从来不会离开器件，不需要采取比特流加密技术。

4.3 远程更新

通过远程使用补丁或者升级包对现场 FPGA 的比特流进行升级所带来的安全问题类似于在联网的个人计算机中使用软件升级包带来的安全问题。很多安全相关的问题都是类似的，例如重放（replay）攻击和中间人（man-in-the-midelle）攻击。

4.3.1 认证

在软件领域中，能够控制一个组织的软件更新机制的攻击者可以在大量的系统

中植入恶意软件。如果攻击者能够重配置、关闭或者破坏用作网络路由器的 FPGA，也可能发动拒绝服务（DoS）攻击。尽管 FPGA 支持本地编程，但还是有一些 FPGA 通过网络进行远程更新。具有远程更新能力的 FPGA 必须设置相关的认证机制，以保证只有具有相关授权的管理员才能够变更 FPGA 的配置，而且更新过程必须安全传输。

认证机制包括密码、生物特征和令牌（在此，我们将其分别称为著名的你知道的事情，你本身的特征，或者你拥有的事物）。由于每种认证机制都有其独特的优点以及局限，在选择一种机制对 FPGA 远程更新管理员进行授权认证时，必须仔细考虑哪一种认证机制能够最好地保护你的系统，防止未授权更新。例如，如果采用了密码认证机制，攻击者可能会利用很多人都会选用容易猜出的密码这一点实施攻击。另外，如果管理员在不同的 FPGA 上使用相同的用户名及密码，则盗用其中的一台 FPGA 就盗用了所有器件。在这种情况下，保证管理员密码的安全就成为防止密码黑客攻击中最重要的一环。另一方面，采用生物特征认证需要管理员实际接触相关的系统，会导致成本的增加，而且必须在误判和漏判之间获得正确的平衡，同时还应该找出重放攻击漏洞。

设计提示：密码。用户、开发者以及管理员必须选择具有足够信息量的密码，同时必须要求他们定期更换密码。用户账户必须得到正确管理，同时，不正常的操作行为，例如多次登录失败，应该获取适当的响应。

设计提示：认证。远程更新的认证机制需要仔细考虑。应该综合考虑多种认证机制，在其间进行分析和权衡。在成本确定的情况下，采用哪种认证方式才能够保证最具有成本效率？系统的设计、实施、测试、开发、配置、操作、维护以及审查要求是什么？

4.3.2　可信恢复

对于需要保护高价值数据的系统而言，必须对远程更新的好处和风险进行权衡。如果系统会被针对此更新机制的攻击所盗用，则必须实施可信的恢复机制将系统恢复到安全状态。同时在此过程中，还必须采取及时有效的管理方式，保证关键服务不会受到影响。

4.4　部分可重构

部分 FPGA 能够在运行过程中改变其部分配置。这种动态部分可重构（也称为

部分可重构）的能力可以让器件中某个核的逻辑功能被完全不同的核的逻辑功能所替换。这种节省空间的特征在面积有限的设计中非常有用。

4.4.1 部分可重构的应用

由于对核进行置换操作会增加系统的复杂度和设计成本，因此应用并不广泛。摩尔定律决定了每两年芯片密度就要增加一倍。尽管如此，还是存在一些例外情况，例如可重构纵横开关[11]。在一片芯片上嵌入 N 个节点互相连接的全交叉纵横开关所需要的面积和 N^2 成正比。然而，如果在特定时间内，只有部分节点需要互相连接，则可以采用动态可重构技术，在较小的面积上实现全交叉纵横开关功能。

> 设计提示：部分可重构。总体而言，部分可重构会增加设计的复杂度，使得系统的安全分析更加富有挑战性。

4.4.2 热置换和停机置换的比较

使用了部分可重构技术的系统可以分为两类：一类在进行核置换过程中需要系统停止运行；另一类在置换过程中不需要停止系统运行。第二类系统称为热置换系统。对设计 FPGA 芯片的工程师而言，热置换系统更加困难，因为涉及将原有数据和状态转移到新核中的问题。在典型情况下，热置换系统需要两个核：一个为静态核；另一个为动态核，在静态核更新过程中保持运行。在更新完成之后，系统立即将静态核和动态核进行互换。

4.4.3 内部配置访问端口

部分可重构技术需要使用内部配置访问端口（ICAP）。FPGA 需要从 ICAP 中读出一帧或者一行配置信息。这些配置信息中的一部分需要在被重新写入 ICAP 中之前进行修改。这个过程可以由一个处理器核驱动。非常明显，攻击者可以只通过读取 ICAP 即可获得完整的比特流。因此，绝大多数的 FPGA 在使用部分可重构功能时都禁用了比特流解密机制，以防止攻击者清晰地读出比特流信息。在处理高价值数据的系统的设计过程中，设计师必须仔细考虑这些机制的细节情况。

> 设计提示：内部配置访问端口（ICAP）。尽管启用部分可重构功能的过程中禁用比特流解密机制能够防止加密比特流通过 ICAP 接口失窃，但可以将可重构加密核与 ICAP 接口结合使用来保护比特流的安全[6]。

4.4.4 动态安全性和复杂度

很明显，部分可重构技术让工程分析变得更加复杂。另外，安全分析也变得更加富有挑战性，这主要是由于部分可重构技术增加了系统的复杂度，尤其是这种技术被用于更改系统安全策略时。构建一个能够执行单一静态策略的系统已经够难了，构建一个能够变更其策略的系统无疑将更加困难。在动态安全设计过程中需要考虑下列关键因素：

1）系统可以从中选择的策略的数量。

2）策略是在运行时临时确定的，还是预先确定并通过预先验证的 。

3）策略变更的频率。

4）系统能否恢复到之前的安全策略或者安全状态。

5）系统是否总是单调过渡到更加不严格的策略上。

> 设计提示：混合策略。如果在策略变更之后，某个核对于残留数据和状态不再具有访问权限，则应该对其进行合理的禁用处理。如果系统要求动态安全性，则必须保证只有可信的模块才有权执行策略变更。其他需要考虑的关键因素包括：
>
> 1）系统可以选择的策略数量（应该对此数量进行限制以降低系统复杂度）。应该进行重要性分析以确保系统处于安全状态。随着系统可用的策略数量的增加，分析的次数也随之增加。
>
> 2）所采用的策略是否是预先确定的策略（优选预先确定的策略以便简化安全分析）。
>
> 3）策略变更的频率（频率越低越好，因为存在隐蔽信道风险）。
>
> 4）系统是否能够恢复到之前的策略（不能恢复更好，这样可以有效防止隐蔽信道风险）。
>
> 5）系统是否总是单调过渡到更加不严格的策略上（是这样更好，因为存在数据残余风险）。

4.4.5 客体复用

如果系统在进行策略变更之后采用了更加严格的安全策略，则必须对系统的存储器以及状态进行擦除操作，以清除其中的非法数据残余。安全客体复用是指将系统资源重新安全地配置给处理单元（例如，某个核），配置方式必须能够防止新的处理单元窃取无意留置在回收资源中的残余信息。

在支持部分可重构的 FPGA 中，设计师不希望之前核残余的数据被继续使用，

因为这可能会在无意间导致可供攻击者利用的漏洞。在参考文献［7］中讲述了可以使用 MicorBlaze 软核处理器通过 Xilinx ICAP 接口一次读取一帧的配置比特流。在读取一帧比特流数据之后，此帧的部分信息被擦除，修改后的新的数据流被重新写入到器件中。这个过程可以通过多种器件配合高效完成。如果 ICAP 的速度为50MHz，则单帧数据可以在大约 10ms 内完成重构，具体时间取决于器件的规模。总重构时间和需要重构的帧数量成正比。

在安全客体复用失败的情况下，会发生数据残余现象。导致此现象的原因包括存储设备的物理特性，或者删除操作不彻底等。例如，即使在室温条件下，动态随机访问存储器（DRAM）在电源中断之后，还能将其存储状态保持数秒钟甚至数分钟的时间。在冷启动攻击情况下，采用压缩空气将 DRAM 冻结，之后可以将其从主板上拆除，并安装到攻击者计算机的主板上，攻击者就可以读取其中存储的信息（针对这种物理攻击方式需要采取防篡改措施，例如线网）[5]。由于删除不彻底导致数据残余的一个例子就是文件的删除。即使在删除后，文件可能依然保留在硬盘上，直到其被新的文件覆盖为止。如果恶意攻击者获得相关的存储设备，就有可能从中恢复相关的敏感信息。用于处理磁盘中含有的残余数据的方法包括多次重写覆盖、消磁、加密，以及物理摧毁相关磁盘。

FPGA 中的数据残余取决于系统的设计方式以及具体的数据，因为数据 0 和数据 1 会以不同的速度加载到器件中[16]。数据残余还取决于比特位与逻辑模块相关还是与内部线路相关，这两种类型的数据存储具有不同的供电电压。正如上述的冷启动例子一样，FPGA 中的数据在低温条件下会保留更长的时间。即使在中断电源之后，FPGA 依然可以将其中的数据保留 2min 左右。对残余的数位进行分析会让攻击者获知纯文本比特流相关信息。通过将电源供应装置接地，可以降低数据残留时间，在 1ms 内即可以实现 20% 的信息删除[12]。

4.4.6 完整性验证

Benjamin 等人开发了一种利用部分可重构技术在运行时检查设计完整性的方法[4]。他们采用称为增益散列的一种特殊形式的加密散列方法[3]，来保证 FPGA 的配置绝对不会进入非法状态。增益散列能够解决如下问题：假设 Alice 希望计算一本 1000 页的书的散列值。而 Alice 的加密散列函数会将整本书作为一个输入以便完成计算。现在假设 Alice 只改变了书的一页。增益散列就会减去之前那页的散列值，再加上修改后那一页的散列值。计算得到的散列值等于修改后的书的散列值，但 Alice 就不需要再次阅读其余的 999 页。图 4-1 中说明了增益散列过程。尽管在银行中这种方法不是特别有用，但还是有部分例子说明其可以用于计算机结构中，包括用于安全处理器的存储器完整性验证过程中[14]。Glas 等人减去了被置换出去的核的散列值，再加上置换进来的核的散列值，确保计算得到的

散列值总是属于已知的良好散列值的子集。如果配置进入非法状态，则系统将会做出响应。

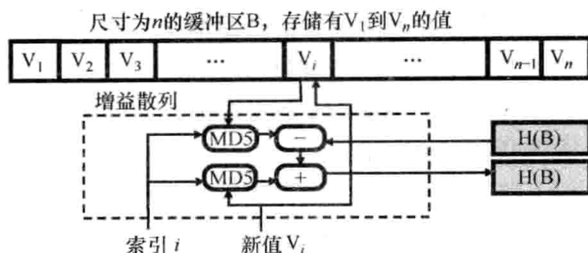


图 4-1　安全增益散列是一种它在缓冲区修改后不需要重新检查完整缓冲区域。保持任意长缓冲区 H（B）加密安全散列值的方法。修改同时作用在散列和缓冲区域上，代替它的是将原始的散列值从 H(B) 中减去，再加上由于修改后的散列值，即可以得到新的散列值 H(B′)

设计提示：动态比特流完整性验证。加密散列函数可以用作运行时比特流，或者用作部分比特流完整性验证的基础工具。实现此功能的一种方法是将 MicroBlaze 软核处理器和部分可重构功能结合使用，来驱动这个过程。每次从 ICAP 读出一帧配置位，之后针对部分或者所有比特流计算其散列值。由于使用部分可重构技术将禁用比特流解密机制，因此可能需要自行执行比特流解密操作[6]。在实施之前，必须仔细考虑这么做增加的系统的复杂度，是否值得相关投入成本，是否会使设计暴露在更大的风险中，攻击者需要采取何种方式才能攻破这种完整性验证机制。

参 考 文 献

1. R. Anderson, M. Kuhn, Tamper resistance: a cautionary note, in *Proceedings of the Second USENIX Workshop on Electronic Commerce*, Oakland, CA, November 1996
2. K. Austin, Data security arrangements for semiconductor programmable devices. US Patent 5,388,157, February 1995
3. M. Bellare, D. Micciancio, A new paradigm for collision-free hashing: incrementality at reduced cost, in *Proceedings of Eurocrypt'97*, Konstanz, Germany, May 1997
4. B. Glas, A. Klimm, O. Sander, K. Müller-Glaser, J. Becker, A system architecture for reconfigurable trusted platforms, in *Proceedings of the 2008 Conference on Design Automation and Test in Europe (DATE'08)*, Munich, Germany, March 2008
5. J.A. Halderman, S.D. Schoen, N. Heninger, W. Clarkson, W. Paul, J.A. Calandrino, A.J. Feldman, J. Appelbaum, E.W. Felten, Lest we remember: cold boot attacks on encryption keys, in *Usenix Security Symposium*, San Jose, CA, July 2008
6. S. Harper, R. Fong, P. Athanas, A versatile framework for FPGA field updates: an application of partial self-reconfiguration, in *Proceedings of the 14th IEEE International Workshop on Rapid System Prototyping*, June 2003

7. T. Huffmire, B. Brotherton, G. Wang, T. Sherwood, R. Kastner, Moats drawbridges: an isolation primitive for reconfigurable hardware based systems, in *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2007
8. A.B. Kahng, J. Lach, W.H. Mangione-Smith, S. Mantik, I.L. Markov, M. Potkonjak, P. Tucker, H. Wang, G. Wolfe, Constraint-based watermarking techniques for design IP protection. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. **20**(10), 1236–1252 (2001)
9. T. Kean, Secure configuration of field programmable gate arrays, in *Proceedings of the 11th International Conference on Field Programmable Logic and Applications (FPL'01)*, Belfast, UK, August 2001
10. P. Lysaght, D. Levi, Of gates and wires, in *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, Santa Fe, NM, April 2004
11. E. Simpson, P. Schaumont, Offline HW/SW authentication for reconfigurable platforms, in *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, Lausanne, Switzerland, September 2006
12. S. Skorobogatov, Low temperature data remanence in static RAM, Cambridge University Technical Report UCAM-CL-TR-536, ISSN 1476-2986, June 2002
13. G.E. Suh, S. Devadas, Physical unclonable functions for device authentication and secret key generation, in *Design Automation Conference (DAC)*, San Diego, CA, June 2007
14. G.E. Suh, B. Gassend, M. van Dijk, S. Devedas, Efficient memory integrity verification and encryption for secure processors, in *Proceedings of the 36th Annual International Symposium on Microarchitecture (MICRO-36)*, San Diego, CA, December 2003
15. S. Trimberger, Method and apparatus for protecting proprietary configuration data for programmable logic. US Patent 6,654,889, 2003
16. T. Tuan, T. Strader, S. Trimberger, Analysis of data remanence in a 90 nm FPGA, in *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC)*, San Jose, CA, September 2007

第 5 章　FPGA 的存储保护

摘要： 本章以形式化的正规语言为基础，对存储访问策略语言（Huffmire 等人，《Proceedings of the European Symposium on Research in Computer Security》，2006 年 9 月于德国汉堡）进行了说明，并阐述了该语言对包括隔离、受控共享和中国墙在内的高级安全策略的描述。本章还描述了一个策略编译器，该编译器可以把用访问策略语言描述的存储访问策略编译为可综合的硬件模块（Huffmire 等人，《Proceedings of the European Symposium on Research in Computer Security》，2006 年 9 月于德国汉堡）。

5.1　概述

必须将外部资源（如片外 DRAM）的管理与 IP 核本身分开，这一点对 FPGA 上的 IP 核至关重要。虽然通用处理器系统通常具备可提供存储保护的虚拟存储机制，但是 FPGA 的物理地址空间和程序结构通常较为扁平，缺乏控制支持（例如来自操作系统的支持）。这一情况可能导致各 IP 核之间由于存储器的读写产生相互干扰，这种读写可能是因为出错，也可能是出于恶意。为避免这一问题的发生，必须引入所有 IP 核都应遵守的“存储访问策略”。可以利用 FPGA 的可重构性来建立实施该访问策略的机制。

存储访问策略描述了哪些存储器访问是合法的，哪些是非法的（有的访问策略采取消极方法，有的采取积极方法），并使用特定的语言对该访问策略进行形式化描述。具有坚实形式化基础的描述使关于访问策略完整性的推论得以实现，并能自动地改进访问策略。通过一套工具能够把策略描述自动地转换并综合成为监视器电路。综合产生的监视器比特流与其他核一起加载到 FPGA 上，通过监视每个核的每一次存储访问来执行存储访问策略，并根据该策略判断允许还是禁止访问。

本章提出的存储保护技术的目标是为构建可靠的 FPGA 系统制定具有凝聚力的策略。为了嵌入式设计社团能够接受该方法论，其设计流程必须能够生成用 FPGA 结构实现的高性能电路。为了符合安全社团的要求，所产生的逻辑还必须忠实可靠地执行安全策略。最后，上述两个社团必须都能理解并同意使用该技术。

设计提示："实现安全策略的硬件机制"。要求实现安全策略的机制不仅能在芯片实现的面积和时序上具有很高的效率，而且对整个系统产生的副作用还必须尽可能地小。实现机制必须能根据具体的设计忠实、可靠、合法地执行安全策略，而且该安全策略本身必须正确无误。在存储访问安全性设计过程中，拥有使用方便且容易理解的策略构建工具，对确保策略的正确性十分有用。

5.2　FPGA 上的存储保护

典型的策略应该是，必须防止嵌入式系统的每个核在未经允许的情况下引用或修改属于其他核的存储器。存储器可分为各种不同的种类：片上块式存储器（Block RAM）、片外动态随机存储器（DRAM），以及诸如闪存等片上或片外辅助存储器。各类存储器都需要高效、灵活和受保护的分配与共享机制。在采用通用型处理器的系统中，存储保护通常以页面表和"旁路转换缓存"（TLB）为基础。目前已经引入了一种被称为"超级页面"的超大存储页面来降低旁路转换缓存的未命中率[15]。然而，通过全局属性来实现的单进程存储保护效率很低。与此相反，分段式存储器[18]和细粒度 Mondrian 存储保护[22]将每个进程与同一存储区中的不同权限关联起来。

在采用简单的线性寻址的现代 FPGA 上，存储器并未受到硬件机制的保护。即使是旁路转换缓存这样一种基本的保护方式也很难在嵌入式处理器或可配置设备中找到。尽管旁路转换缓存可以加快对页面表的访问速度，但却是以牺牲系统性能和耗用更多相关存储器为代价的。

存储保护对于防止错误和安全攻击都十分重要。不幸的是，在软件中对存储器进行管理并不那么简单，某些细微的存储错误可能很难被察觉。为多个并发式"硬件"模块提供存储保护需要使用一种不同的方法，该方法借鉴了分离内核中某些关键性的想法。分离内核[17]使用虚拟化技术分离软件进程，这种分离等同于物理分离。所有资源都被划分为若干个分区，每个分区彼此隔离，除非构建一条明确的信道，才能互相通信。在多层级系统内，每个分区都分配有一个层级，且分区之间的通信必须遵守映射到该分区集合上的 MLS 标示网格[7]的流规则。

我们可以把 IP 核与存储区域作为（算法）内核不同分区的理念应用到 FPGA 上。第 7 章中将要探讨采用"壕沟"技术来隔离各个核，采用"吊桥"技术来安全地控制各个核之间的交互[10]。为了对每一次的存储访问都进行有效检验，可配置的硬件内部设置了一种可以识别合法访问语言的引用监视器，所有对片外存储器

进行的访问都必须通过该引用监视器。将执行模块置于芯片上会比置于一块单独的片外硬件模块上所产生的延迟要低。除了对核进行保护外，壕沟与吊桥技术还能实现执行模块的隔离，而且还能防止绕开执行模块。

5.3 策略描述与综合

尽管典型的可重构式系统缺乏标准的存储保护机制，但是依然可以利用 FPGA 精细的可编程属性将高效的存储保护机制融合进来。编译器会将存储访问策略直接转换为可以提供状态化的字级存储保护电路。

本节将对存储访问策略语言进行说明，并展示如何表达一项策略并将其编译为对硬件模块的可综合描述。除了对“形式化的顶层规范”（FTLS）进行描述外，本节还会对将策略转化为执行模块的设计流程进行说明。

5.3.1 存储访问策略

存储访问引用验证机制的形式化顶层规范是以系统要求和组织安全策略为基础的[21]。本章所描述的执行机制为执行机制的执行监视（EM）类成员[19]。该类成员负责监视某个目标的执行情况。在 FPGA 上，该目标为所有核的集合，如图 5-1 中的范例所示。由于该执行机制是一种引用验证机制（RVM），因此它必须受到保护，不允许绕过，并且可以验证[3]。

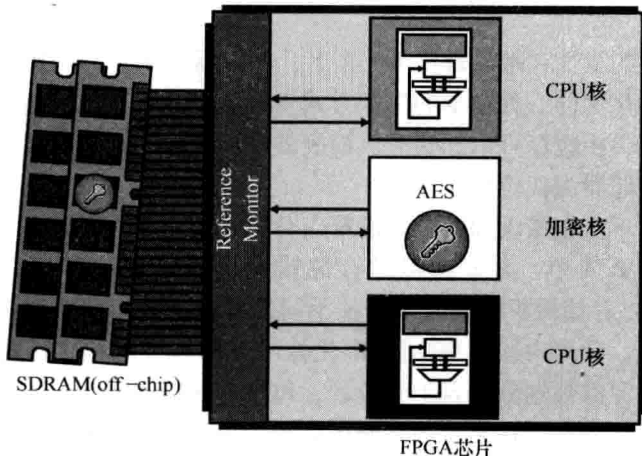


图 5-1 拥有两颗 CPU 和一颗 AES 加密核的 FPGA，以及置于 FPGA 之上的引用验证机制（RVM）

形式化顶层规范精确地描述了一系列有待识别的合法存储访问模式。该引用监视器必须能够跟踪任意大小的存储区，只允许合法的访问，而禁止一切其他访问。

形式化顶层规范所使用的语言必须能够清晰地描述经典安全策略，包括隔离和受控共享等。工程师可以使用该语言表达策略的内容，并使用编译程序来将其转换为正则表达式。采用正规语言可以提供一种可读性强的策略表达方式。现有技术可将正则表达式转化为状态机和硬件[1]，从而将高级规范自动转换为可执行的电路。

我们已经开发出两种meta语言：一种低级meta语言和一种高级meta语言。在我们推荐使用的低级meta语言中，“访问模块（M）”是芯片上的各个核的标识。在安全术语中，模块还可以被称为主体或主干。“访问方法（A）”用于描述权限，即一个模块能够对客体实施的操作，如读取、写入、清零等。在低级语言中，存储器被分为若干个地址范围，这些地址范围都是操作的对象。“存储区说明符”（R）是可为其分配离散式权限值的存储区的集合。其他范围则被保留（如专供RVM使用的范围）。低级语言使用符合语法的“语句结构”来描述一项访问策略，且每一结构都指定了各个模块（M）、访问权限（A）和范围（R）之间的关系，其中箭头（→）左侧的元素会被右侧的表达式所代替。

低级语言使用“存储访问描述符”来指定一个模块对某个范围的访问权。这些描述符是低级语言的语句结构的“终结符”，并且是（M，A，R）形式的元组。虽然存储访问（M，A，k）涉及一个单一的地址k，但可以为每个地址范围都定义一个存储访问描述符（M′，A′，R）。当且仅当M＝M′，A＝A′，且$R_{low} \leqslant k \leqslant R_{high}$时，（M，A，k）才包含在（M′，A′，R）之中。

所有可能的描述符元组的空间为交叉乘积$\Sigma = M \times A \times R$。前面提到的存储访问策略定义成为一种“正规”语言，$L \subseteq \Sigma$。

该语言定义了一个可能的描述符的子集，这些描述符可能是有限的，也可能是无限的。在FPGA上，执行策略需要精确定义一种表述“L”的策略表示法，一种能够自动生成识别合法存储访问序列的电路的方式，以及一种防止非法访问的方式。

一项隔离策略的简单实例有助于描述上述策略表示法。假定一个系统设计拥有两个模块，且每个模块只可访问其自身的特定范围内的存储器。以下语句结构描述了使用该低级语言来实现该方案的方法：

$rw \rightarrow r \mid w;$

$Range_1 \rightarrow [0x8e7b008, 0x8e7b00f];$

$Range_2 \rightarrow [0x8e7b018, 0x8e7b01b];$

$Access_1 \rightarrow \{Module_1, rw, Range_1\};$

$Access_2 \rightarrow \{Module_2, rw, Range_2\};$

$Policy \rightarrow (Access_1 \mid Access_2)^*;$

Policy是一个以$Access_1$和$Access_2$的方式来定义访问策略（L）的非终结符，在前面的语句结构中，依次对$Access_1$和$Access_2$进行了定义。尽管在上述实例中，

$Access_1$ 和 $Access_2$ 非常简单，使用形式化语法却能够以一种层次化的方式来精确编写更为复杂的策略，实现更为复杂的存储访问集合。

低级语言必须正规，以便其语句（即策略所允许的那些行为）能够为 FSA 所识别[14]，从而在硬件中得到有效实现。为了简化策略的建立工作，并不要求策略采用右线性方式或左线性方式来表达。编译器仅仅将策略从广义的正规语法转化为严格的正规语法，以便能够方便地表达范围的概念。

地址范围可以具有任意大小，从一个字的执行单元到整个地址空间均可。这有助于建立可分享的“控制字”并能够为各个模块所使用，以实现安全可靠的协调。范围不允许重叠，通过静态检查来确认这一点。尽管低级语言朴素而简单，但它灵活多变、通用性强，能够表达各种各样的经典安全策略，包括隔离和中国墙。以上是对策略定义格式的描述，下一节将展示如何将存储访问策略自动被转换为高效的可重构硬件模块。

设计提示：构建一项策略。策略构建的第一步是指定范围。接下来，是指定访问描述符。每个 $Access_i$ 能够指定一个或多个访问描述符，多个描述符之间用“逻辑或”符号（|）隔开。最后，根据指定存储访问描述符来指定策略。只具有一种状态的策略（即“无状态”策略）可以简单书写为 $(Access_1 \mid Access_2 \mid Access_3 \mid \cdots \mid Access_n)^*$。具有一种以上状态的“状态化”策略需要指定触发事件（能为该语言所识别的特殊终结符），该类触发事件能够实现从一种状态到另一种状态的转换。本章稍后将介绍状态化策略和无状态策略的几个实例。

5.3.2 硬件综合

策略编译程序能够将一项访问策略的语法转换为电路，该电路会被加载到 FPGA上，以便在运行期间执行该策略。该电路可分为两部分：确认某一给定的地址所属范围的范围检测硬件和识别语言的状态机。设计流程由以下步骤组成：

1）工程师编写访问策略，作为编译器的输入。

2）编译器执行下列动作：

①根据该策略构建一棵语法树；

②将该语法树转换为一种扩展后的中间形式；

③将“策略”扩充为一种正则表达式；

④将该正则表达式转换为非确定性有限自动机（NFA）；

⑤将该 NFA 转换为最小化确定性有限自动机（DFA）；

⑥将每个范围都转换为对齐到 2 的次幂的覆盖集合；

⑦将范围检测与状态机逻辑作为可综合的Verilog输出；

3）使用商业化的综合工具（如Altera Quartus软件）将Verilog硬件描述作为输入进行电路综合、布局和布线。

4）使用比特流下载程序将该比特流下载到FPGA上。

5）在运行过程中，如果与主体请求相对应的“描述符”为DFA所接受，则该要求会得到允许。

5.3.2.1 设计流程的细节

1. 访问策略

下面说明一项简单的策略是如何被转换为电路的。之前所描述的隔离策略具有两个模块，且每个模块只能访问其各自的范围。可以用以下方法更为简洁地表达同样的策略：

Access→{Module_1, rw, Range_1} | {Module_2, rw, Range_2}；

Policy→(Access)*；

2. 分析树的构建与转换

然后，编译器会使用Lex[13]和Yacc[12]来根据该策略构建一棵分析树，如图5-2所示。

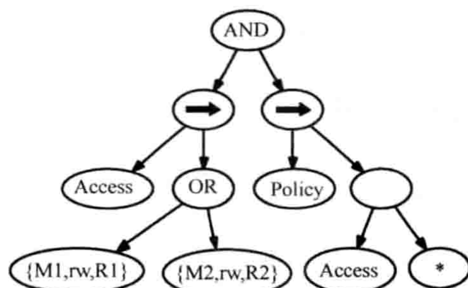


图5-2 简单策略的分析树（“AND”为根节点，其两个子节点中的每一个都是与该策略的两条规则相对应的一棵子树。“右箭头符号”是所有这些子树的根节点）

接下来，编译器会使用每个语句结构右边的内容来替换左边的内容，如此交迭反复地替换掉所有的非终结符。图5-3展示的是转换后的分析树，编译器根据该分析树能够生成一条正则表达式。

3. 生成正则表达式

随后，编译器会横贯与“策略”相对应的子树来生成正则表达式。在该简单实例中，正则表达式很容易就能直接根据语法生成，而更为复杂的策略则需要精心构建扩展分析树。

(({Module_1, rw, Range_1}) | ({Module_2, rw, Range_2}))*

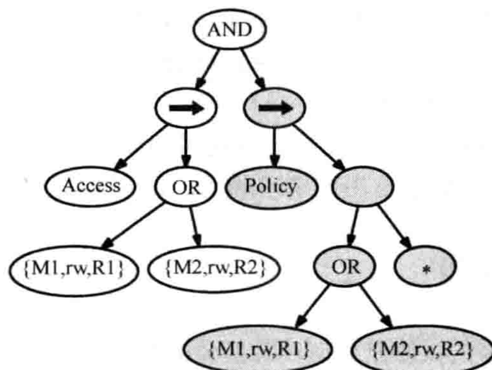


图 5-3　扩展分析树（阴影部分是与该策略中的第二条规则相对应的“右子树”，而“Access”（访问）已被替换为 {$Module_1$，rw，$Range_1$} | {$Module_2$，rw，$Range_2$}）

4. NFA 的构建

然后，编译器会使用 Thompson 算法[1]来根据正则表达式构建 NFA。图 5-4 展示了该策略的 NFA。

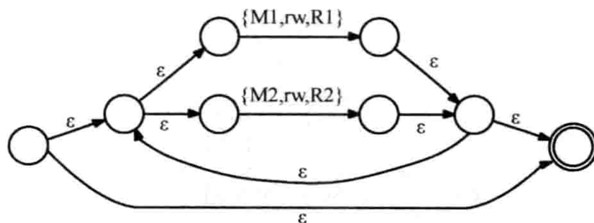


图 5-4　正则表达式衍生出的 NFA（“OR”需要有四个 Epsilon 转换过程顶部中央位置，最靠近 {M1，rw，R1} 和 {M2，rw，R2}）。其他 4 个 Epsilon 转换过程是为 Kleene star 的运行而做的）

5. 将 NFA 转换为 DFA

接下来，编译器将使用子集构造来将 NFA 转换为 DFA[1]。然后，编译器将采用 Grail[16]实现的 Hopcroft 的分区算法[1]将 DFA 最小化。图 5-5 展示了该策略的最小化 DFA。

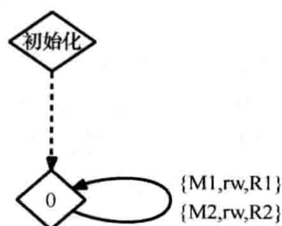


图 5-5　将 NFA 转换为最小化的 DFA（由于该 DFA 只有一种状态，因此它所执行的策略被称为“无状态”策略）

6. 范围的处理

编译器必须将各个地址范围进行处理，即将其转换为某种格式，允许电路高效地计算包含给定地址的范围，并行搜索整个范围集合。使用“随意”位时，仅最重要的位才

需要检查。例如，10XX与1000、1001、1010和1011或区间[8，11]相对应。要检查“9”（1001）是否处在该范围内，硬件会对1001和10XX的头两位进行XOR的位运算。由于10XOR10的结果为00，因此“9”肯定处在区间[8，11]之内。

大小为2的N次幂的任何范围都可以使用“随意”位来进行描述。对于大小不为2的N次幂的范围，编译器需要将其转换成大小为2的N次幂的O（log2 | range | ）范围的覆盖集合，其中 | range | 为范围内的地址的数目。例如，范围[7，12]（0111，1000，1001，1010，1011，1100）可以被转换为以下范围的集合：{[7，7]，[8，11]，[12，12]}（or {0111 | 10XX | 1100}）。

7. 将DFA转换为Verilog

因为状态机是非常通用的硬件原语，因此有很多成熟的方法来将DFA转换成Verilog等硬件描述语言。图5-6展示了执行模块的结构。输入值为模块的识别码(ID)、操作码和存储访问的地址，输出值只有一位：“1”（同意）或“0”（拒绝）。首先，硬件并行检查所有范围来确定与输入的地址相对应的地址范围。然后，由DFA来处理请求。如果DFA转换为同意状态，则输出值为“1”；如果DFA转换为拒绝状态，则输出值为“0”。如果DFA未识别出该访问请求，则DFA转换为拒绝状态，防止恶意攻击使用未定义的输入值而将输出值置为“1”。

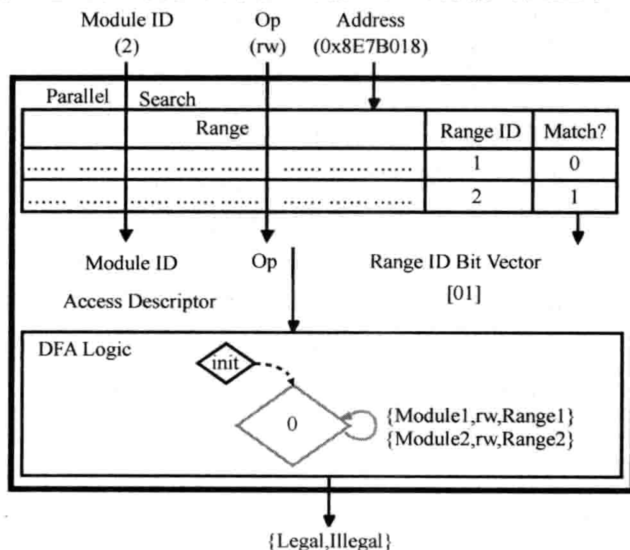


图5-6 执行模块可接受三种输入值：模块ID、操作码和地址。通过并行检索可以确定范围ID。模块ID、操作码和范围ID位矢量共同组成了访问描述符，状态机使用该描述符来确定输出值：“1”（同意）或“0”（拒绝）

8. 状态机综合

最后，将描述执行模块的 Verilog 代码转化为比特流，连同其他核一起加载到 FPGA 上。使商业化综合工具（如 Altera Quartus 设计工具等）综合、优化和实施布局布线。需要进行测试来检验电路的正确运行。

5.4 高级描述语言

策略执行机制如引用监视器等应当与其执行的策略同样出色。5.3.1 节中所描述的 meta 语言属于较为低级的语言，需要嵌入式系统设计人员仔细考虑从模块和范围角度来声明的正则表达式。而高级语言则能够通过高级的安全概念来表达各种策略，降低策略构建过程中人为失误的风险，从而提供更为出色的可用性。

研究表明，可用性对于系统安全性至关重要[23]。安全策略可以在不同的抽象层次上进行表达[21]。处于最高级上的是组织安全策略[20]，这是一份阐述组织机构安全需求的一份书面文件。另一方面，计算机系统具有对更低抽象级别的安全策略的规范以及在更低抽象水平上表达的安全策略的执行机制。目前，确保从高水平的组织安全策略到低水平的实施过程的可靠转换是一个公开的问题（对于这一问题的技术已在 2.6.7.3 小节加以探讨）。

表达存储访问策略的高级语言增强了设计流程的可用性。在这一方法中，处于中间位置的编译器将采用这种高级语言表达的策略转换为本章之前所述的低级语言。采用该高级语言表达的策略复杂性大大降低，尤其是中国墙、高水位线、低水位线等策略参数（比如利益冲突类的数目、安全标识空间的大小等）呈指数性增长。用高级语言表达高水位线或低水位线策略只需要指定每个模块和范围的安全标识，而表达中国墙策略则只需要指定每个利益冲突类的元素即可。虽然目前实现的编译器已经由开发人员进行了一些测试，但还需要进行进一步的系统测试以便让该编译器达到生产级的水准。换而言之，实现的编译器还存在着一些漏洞，在设计工作中使用它之前，我们应当对每个工具的输出结果进行检查。具体而言，在将电路应用到设计中之前，我们应当对电路进行仿真，输入一组测试参数，验证相应的输出结果。该高级语言支持以各种策略：

1）隔离策略。

2）受控共享策略。

3）访问列表策略。

4）中国墙策略。

5）编辑策略。

6）Bell 和 LaPadula 策略。

7）高水位线策略。

8）Biba策略。

9）低水位线策略。

示例隔离策略的表达如下：

Isolation

$Compartment_1 \rightarrow Module_1$;

$Compartment_1 \rightarrow Range_1$;

$Compartment_2 \rightarrow Module_2$;

$Compartment_2 \rightarrow Range_2$;

注意在文件的第一行指定了策略的类型。在高级语言中，工程师指定一套分隔区，每个分隔区包括一个或多个模块以及一个或多个范围。具有多个模块的分隔区是一个等价类，该等价类的所有元素都会得到策略方面的同等对待。

5.5 示例策略

由于已经展示了隔离策略，本节将展示如何表达其他类型的策略，包括涉及撤销或条件性访问的“状态性”策略。要了解更多实例，请参阅参考文献［11］和［9］。

5.5.1 受控共享

尽管隔离策略可防止核之间非预期信息流的传递，但有时各个核需要能够彼此通信。高级和低级策略语言能够指定某个核直接向另一个核安全地传送数据而无需中间通信缓冲或该数据的多份复制。相反，对数据指定范围的访问权可以以某种方式从一个核传递给下一个核，改方式能够防止对该数据的同时访问或部分传送。例如，如果 $Module_1$ 希望安全地将数据传送给 $Module_2$，那么访问策略则能够同步对共享缓冲区访问权的转换。该策略用低级语言描述如下：

$Access_1 \rightarrow \{Module_1, rw, Range_1 | Range_3\} | \{Module_2, rw, Range_2\}$;

$Access_2 \rightarrow \{Module_1, rw, Range_1\} | \{Module_2, rw, Range_2 | Range_3\}$;

$Trigger \rightarrow \{Module_1, rw, Range_4\}$;

$Policy \rightarrow (Access_1)^* (\varepsilon | Trigger(Access_2)^*)$;

> 设计提示：使用Epsilon术语的正则表达式。为状态性策略指定“策略”表达式更为精妙。由于该策略不太可能转移（触发）到 $Access_2$ 中，而会乐意永远处于 $Access_1$ 当中，因此需要使用代表空字符串的 ε 术语。

首先，$Module_1$ 能够访问 $Range_1$ 和 $Range_3$（交换缓冲），但是 $Module_2$ 只能访

问 $Range_2$。$Module_1$ 在访问 $Range_4$（控制字）时，会向监视器表明它已做好了权限转移的准备。该触发器事件会停用 $Access_1$，取消 $Module_1$ 对 $Range_3$ 的访问权。该触发器事件还会授予 $Module_2$ 对 $Range_3$ 的独占访问权。

在高级语言中，工程师会指定“from”模块、“to”模块、交换“缓冲区”和“控制字”。由于受控共享已经在隔离环境下得到实施，工程师还要指定分隔区。高级语言“CS”策略的语义是，在“from”模块访问控制字之前，只有它才能访问缓冲区，而在这之后，只有“to”模块才能访问缓冲区（在两种情况下的“rw”模式中）：

CS;

From→$Module_1$;

To→$Module_2$;

Buffer→$Range_3$;

ControlWord→$Range_4$;

$Compartment_1$→$Module_1$;

$Compartment_1$→$Range_1$;

$Compartment_2$→$Module_2$;

$Compartment_2$→$Range_2$;

图 5-7 展示了执行受控共享策略的 DFA。

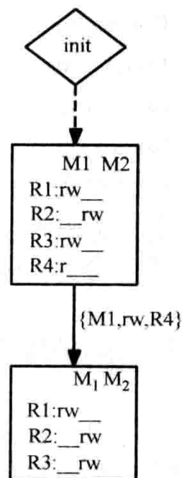


图 5-7　与受控共享策略对应的 DFA

5.5.2　访问列表

有时一长串主体必须访问同一个目标。访问列表策略是一项隔离策略，在该策略中有一个或多个模块属于同一列表。该策略中的主体是依照列表而非个别的模块来进行表述，很像是一个角色或群组。在高级语言中，设计人员首先指定各个模块的列表，然后依照这些列表来表达策略。例如：

AL;

$List_1$→$Module_1$;

$List_1$→$Module_2$;

$List_1$→$Module_3$;

$List_1$→$Module_4$;

$List_2$→$Module_3$;

$List_2$→$Module_4$;

$Compartment_1$→$List_1$;

$Compartment_1$→$Range_1$;

$Compartment_2$→$List_2$;

$Compartment_2 \to Range_2$;

上述高级规范可以转换为下列低级规范：

$Access_1 \to \{Module_1, rw, Range_1\} \mid \{Module_2, rw, Range_1\}$
$\mid \{Module_3, rw, Range_1\} \mid \{Module_4, rw, Range_1\}$;

$Access_2 \to \{Module_3, rw, Range_2\} \mid \{Module_4, rw, Range_2\}$;

$Policy \to (Access_1 \mid Access_2)^*$;

图5-8展示了执行访问列表策略的DFA。

5.5.3 中国墙

另一种可以使用该策略语言来有效表达的安全方案是中国墙[6]。假定 $Company_1$ 和 $Company_2$ 是竞争对手，并且它们属于同一利益冲突（COI）类。如果一位律师看了 $Company_1$ 的档案材料，则他或她不应再看 $Company_2$ 的档案文件。但是，如果 $Company_3$ 属于不同于 $Company_1$ 的另外一个利益冲突类，则该律师可以翻阅 $Company_3$ 的档案。图5-9中展示了描述这一状况的Venn图解。在该示例策略中，$Module_1$ 相当于该律师，而每个范围则相当于一家公司。

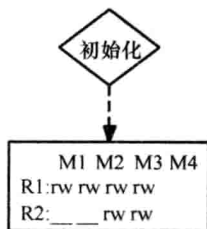


图5-8　与访问列表策略对应的DFA

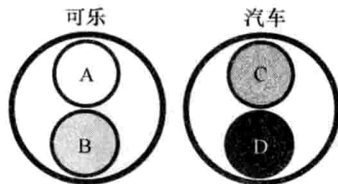


图5-9　该Venn图解展示了两种利益冲突（COI）类，一种是苏打水公司的，一种是汽车公司的。一位在着手A品牌的可乐案例的律师不能着手B品牌的可乐案例，但是可以着手C品牌的汽车案例或D品牌的汽车案例。将其推而广之，如在嵌入式系统中，将A品牌的可乐、B品牌的可乐、C品牌的汽车和D品牌的汽车替换为 $Range_1$、$Range_2$、$Range_3$ 和 $Range_4$

$Access_1 \to \{Module_1, rw, (Range_1 \mid Range_3)\}^*$;

$Access_2 \to \{Module_1, rw, (Range_1 \mid Range_4)\}^*$;

$Access_3 \to \{Module_1, rw, (Range_2 \mid Range_3)\}^*$;

$Access_4 \to \{Module_1, rw, (Range_2 \mid Range_4)\}^*$;

$Policy \to Access_1 \mid Access_2 \mid Access_3 \mid Access_4$;

该中国墙有两个利益冲突类：一个包含 $Range_1$ 和 $Range_2$，另一个包含 $Range_3$ 和 $Range_4$。图5-10展示了执行该策略的DFA。

通过指定属于每个利益冲突类（$Class_1$ 或 $Class_2$）的范围以及作为该策略中的

主体的模块，可以使用高级语言来表达一项中国墙策略，即

Chinese;

$Class_1 \rightarrow Range_1$;

$Class_1 \rightarrow Range_2$;

$Class_2 \rightarrow Range_3$;

$Class_2 \rightarrow Range_4$;

$Subject \rightarrow Module_1$;

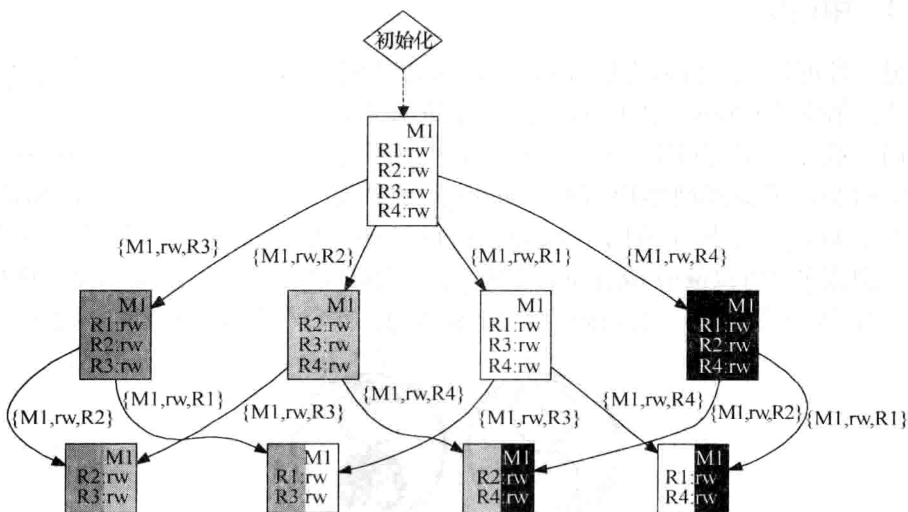


图 5-10　DFA 强制执行中国墙策略。访问过 $Range_1$（白色）的某个核随即就被禁止访问 $Range_2$（浅灰色），但是该核仍旧可以访问 $Range_3$（深灰色）或者 $Range_4$（黑色），因为 $Range_3$ 和 $Range_4$ 与 $Range_1$ 和 $Range_2$ 属于不同利益冲突类

5.5.4　Bell 与 LaPadula 保密模型

Bell 与 LaPadula（B&L）模型是一种多级安全性形式模型。在该模型中，一个主体不可读取拥有较高安全标记的对象（禁止向上读取），也不能写入拥有较低安全标记的对象（禁止向下写入）[4]。该模型在设计上是用于保护分级信息的机密性。假定在标记空间内有四种标记，即绝密（TS）、机密（S）、秘密（C）和非密（U）。设计人员可通过指定每个模块和范围的安全标记，用高级语言来表达 B&L 策略，如

B&L;

$Module_1 \rightarrow TS$;

$Module_2 \to U$;

$Range_1 \to U$;

$Range_2 \to U$;

$Range_3 \to TS$;

上述高级规范可以转换为下列低级规范：

$Policy \to (\{Module_1, r, Range_1\} \mid \{Module_1, r, Range_2\}$
$\mid \{Module_1, rw, Range_3\} \mid \{Module_2, rw, Range_1\}$
$\mid \{Module_2, rw, Range_2\} \mid \{Module_2, w, Range_3\})^*$;

图 5-11 展示了执行 B&L 策略的 DFA。

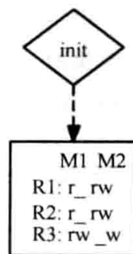


图 5-11　与 B&L 策略对应的 DFA

5.5.5　高水位线

高水位线模型是对 B&L 的扩展。在不允许向上读取方面高水位线与 B&L 相同，但是在高水位线中允许向下写入。在向下写入之后，被写入的对象的安全标记必须更改为实施该写入动作的主体的标记。与 B&L 不同，高水位线策略是状态性策略。设计人员可通过指定每个模块和范围的安全标记来用高级语言表达高水位线策略，如

High;

$Module_1 \to TS$;

$Module_2 \to U$;

$Range_1 \to U$;

$Range_2 \to U$;

$Range_3 \to TS$;

上述高级规范可以转换为下列低级规范：

$Trigger_1 \to \{Module_1, w, Range_1\}$;

$Trigger_2 \to \{Module_1, w, Range_2\}$;

$Access_0 \to (\{Module_1, r, Range_1\} \mid \{Module_1, r, Range_2\}$
$\mid \{Module_1, rw, Range_3\} \mid \{Module_2, rw, Range_1\}$
$\mid \{Module_2, rw, Range_2\} \mid \{Module_2, w, Range_3\})^*$;

$Access_1 \to (\{Module_1, rw, Range_1\} \mid \{Module_1, r, Range_2\}$
$\mid \{Module_1, rw, Range_3\} \mid \{Module_2, w, Range_1\}$
$\mid \{Module_2, rw, Range_2\} \mid \{Module_2, w, Range_3\})^*$;

$Access_{12} \to (\{Module_1, rw, Range_1\} \mid \{Module_1, rw, Range_2\}$
$\mid \{Module_1, rw, Range_3\} \mid \{Module_2, w, Range_1\}$
$\mid \{Module_2, w, Range_2\} \mid \{Module_2, w, Range_3\})^*$;

$$\text{Access}_2 \rightarrow (\{\text{Module}_1, r, \text{Range}_1\} \mid \{\text{Module}_1, rw, \text{Range}_2\}$$
$$\mid \{\text{Module}_1, rw, \text{Range}_3\} \mid \{\text{Module}_2, rw, \text{Range}_1\}$$
$$\mid \{\text{Module}_2, w, \text{Range}_2\} \mid \{\text{Module}_2, w, \text{Range}_3\})^*;$$

$$\text{Access}_{21} \rightarrow \text{Access}_{12};$$

$$\text{Path}_1 \rightarrow (\varepsilon \mid \text{Trigger}_1\ \text{Access}_1{}^*(\varepsilon \mid \text{Trigger}_2\ \text{Access}_{12}{}^*));$$

$$\text{Path}_2 \rightarrow (\varepsilon \mid \text{Trigger}_2\ \text{Access}_2{}^*(\varepsilon \mid \text{Trigger}_1\ \text{Access}_{21}{}^*));$$

$$\text{Policy} \rightarrow \text{Access}_0{}^*(\varepsilon \mid \text{Path}_1 \mid \text{Path}_2);$$

> 设计提示：带有路径算式的正则表达式。这是一个采用“菱形”的状态性策略。该菱形自上而下有两条可能的路径：一条是穿过菱形的右半部分（路径 1），另一条是穿过菱形的左半部分（路径 2）。

图 5-12 展示了执行高水位线策略的 DFA。

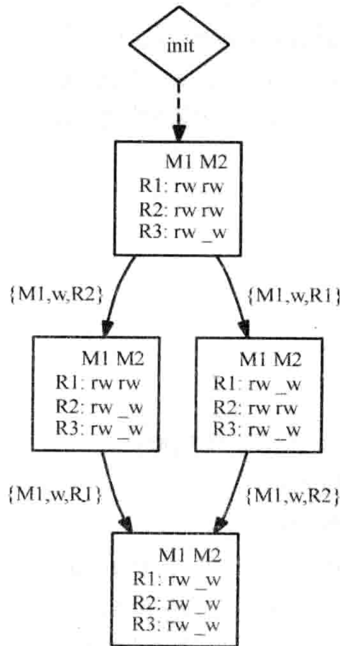


图 5-12　与高水位线策略对应的 DFA

5.5.6　Biba 完整性模型

Biba 模型是 Bell-LaPadula 模型的二元体[5]。由于 Biba 用于保护数据的完整性，因此在该模型中向下读取和向上写入都是禁止的。可以通过指定每个模块和范围的安全标记来用该高级语言来表达一项 Biba 策略，如

biba;

$Module_1 \rightarrow TS$;

$Module_2 \rightarrow U$;

$Range_1 \rightarrow U$;

$Range_2 \rightarrow U$;

$Range_3 \rightarrow TS$;

在这里，TS 为高完整性，而 U 为低完整性。上述高级规范可以转换为下列低级规范：

$Policy \rightarrow (\{Module_1, w, Range_1\} \mid \{Module_1, w, Range_2\}$

$\mid \{Module_1, rw, Range_3\} \mid \{Module_2, rw, Range_1\}$

$\mid \{Module_2, rw, Range_2\} \mid \{Module_2, r, Range_3\})^*$;

图 5-13 展示了执行 Biba 策略的 DFA。

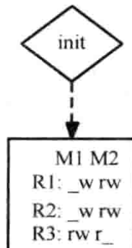


图 5-13　与 Biba 策略对应的 DFA

5.5.7　编辑

编辑是将一份文档中敏感部分删除的过程，从而使具有较低权限的人员也能阅读该文档。图 5-14 所示的拥有三颗核的 FPGA 系统的实例。在该情形中，一个多级数据库同时包含有秘密信息和非密信息。一颗核（$Module_1$）能够读取和写入机密信息，并能够读取非机密信息。另一颗核（$Module_2$）则只允许读取和写入非机密信息，而第三颗核（$Module_3$）则担任受信任的服务器，从数据库中检索信息以便对查询作出响应，必要时进行编辑，并将数据写入到存储器的特定区域内（$Range_3$）。$Module_1$ 和 $Module_2$ 通过向某个控制字（$Module_4$）中写入请求来进行数据库的查询。如果 $Module_1$ 在进行一项数据库查询，则必须临时阻止 $Module_2$ 访问 $Range_3$，以便在响应查询时此处的机密信息能够被写入。$Module_2$ 必须等待受信任的服务器（$Module_3$）将 $Range_3$ 归零。该策略包含两种状态：自由状态，在该状态下 $Module_2$ 能够访问 $Range_3$；限制状态，在该状态下 $Module_2$ 不能访问 $Range_3$。当 $Module_1$ 通过向 $Range_4$ 中写入而执行一项数据库查询时，该触发事件会导致 $Module_2$ 从自由状态变为限制状态。当 $Module_3$ 将 $Range_3$ 归零时，该触发事件会使

$Module_2$ 从限制状态变回到自由状态。在该实例中，已经在隔离环境下进行了编辑，所以仅 $Module_1$ 可以从 $Range_1$ 中读取或向其写入，同时仅 $Module_2$ 可以从 $Range_2$ 中读取或向其写入。

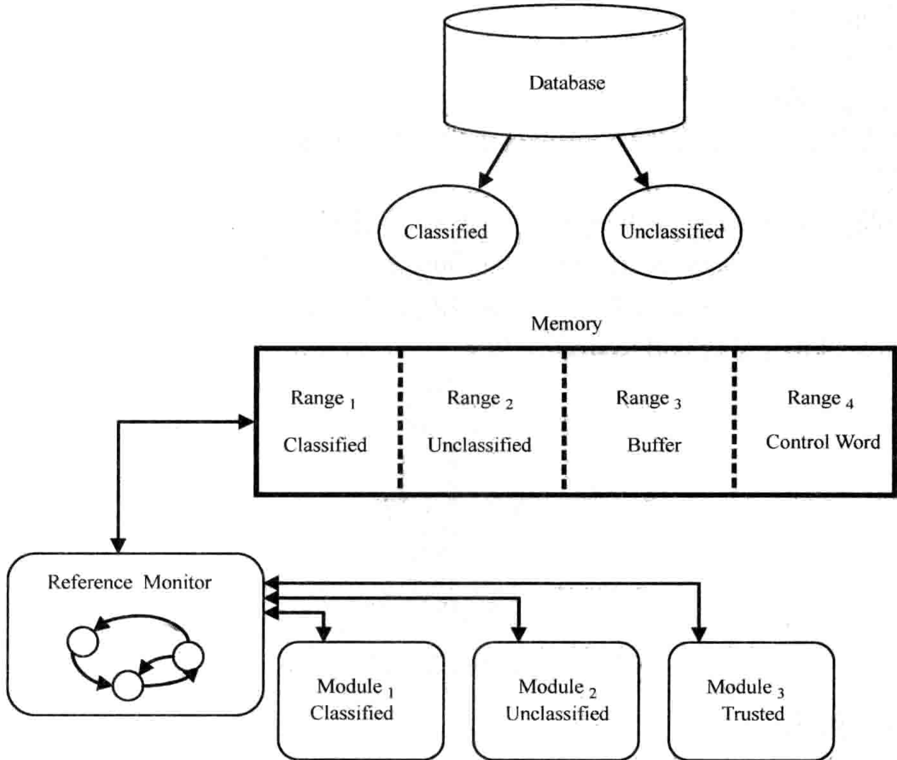


图 5-14　编辑方案的结构

在高级语言中，可以指定自由策略和限制策略，以及触发事件和清除事件。

Redaction;

Restrictive→{$Module_1$, rw, $Range_1$} | {$Module_1$, r, $Range_3$}
　　| {$Module_2$, rw, $Range_2$} | {$Module_2$, w, $Range_4$}
　　| {$Module_3$, rw, $Range_3$};

Liberal→Restrictive | {$Module_2$, r, $Range_3$};

Trigger→{$Module_1$, w, $Range_4$};

Clear→{$Module_3$, z, $Range_3$};

上述高级规范可以转换为下列低级规范：

$Access_1$→{$Module_1$, rw, $Range_1$} | {$Module_1$, r, $Range_3$}
　　| {$Module_2$, rw, $Range_2$} | {$Module_2$, w, $Range_4$}

$\mid \{Module_3, rw, Range_3\};$

$Access_2 \rightarrow Access_1 \mid \{Module_2, r, Range_3\};$

$Trigger \rightarrow \{Module_1, w, Range_4\};$

$Clear \rightarrow \{Module_3, z, Range_3\};$

$SteadyState \rightarrow (Access_1 \mid Clear\ Access_2 * Trigger) *;$

$Policy \rightarrow \varepsilon \mid Access_2 * \mid Access_2 * Trigger\ SteadyState$

$\mid Access_2 * Trigger\ SteadyState\ Clear\ Access_2 *;$

> 设计提示：简化正则表达式。由于该DFA具有循环周期，因此需要使用一条复杂的正则表达式。为了简化该正则表达式，可以在该实例中使用SteadyState等非终结符来代替公用子表达式。

图5-15展示了执行编辑策略的DFA。

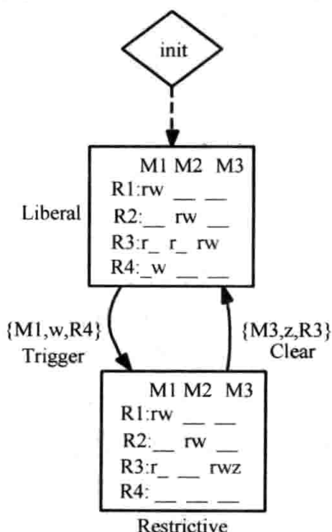


图5-15 与编辑策略对应的DFA

5.6 系统架构

引用监视器必须放置在系统架构内，以便提供不可旁路性和自我保护等特性，并将其对存储系统性能的影响降至最小。系统在核心数量、系统元件的通信方式（通过直接连接、总线或网络）和需保护的资源种类（片上BRAM和SRAM，片外DRAM等）上都会有很大不同。在一个模块通过总线来访问片外DRAM的双核系统中，缺乏经验的设计人员可能会将执行机制放置在总线和存储器之间，从而要求

每一次存储访问前都要等待执行机制的许可。产生的延迟为存储延迟和访问许可时间的总和。而更为高明的做法则是将执行机制安插在总线上，以便对存储器的访问能够与许可的下达同时发生。存储数据的缓冲区在获得访问许可后恢复数据。对于写入操作，该缓冲区会将需写入的数据存储起来，直到获得写入许可。不过这样并不会改善延迟时间和许可时间，除非使用回滚方案。两项策略都提供了必要的隔离，同时，它们还在性能与复杂性之间提供了一种折中方案。

为防止各个模块同时对总线进行访问，必须引入仲裁机制。一种简单的办法是在每个模块和总线之间放置一个仲裁器，这些仲裁器会将每个核对总线的使用限制在其既定的时间片内。本书第 7 章对在总线中融合了仲裁器和引用监视器的多核嵌入式系统进行了说明。该设计实例由两个 MicroBlaze 处理器组成，通过一项状态性策略来管理两个处理器对一个 AES 加密核的共享。

设计提示：执行机制的布置。将策略执行机制布置在什么位置不仅事关安全性，而且也会影响到性能，尤其是在整个系统不得不等待引用监视器的决定才能继续的情况下。而改善这些延迟的技术（如并行性和流水线技术等）又会增加复杂性。

5.7 评估

参考文献［9］和［11］表明，FPGA 引用监视器就面积和性能而言十分高效。电路的复杂度取决于存储区的数目以及 DFA 状态和转换的多少。为了研究范围检测工作对系统性能的影响，通过进行一项实验来改变隔离策略中的存储区的数目，并使用 Altera 的 Quartus 综合工具来综合所生成的执行模块[2]。该实验的结果证明了在电路的大小和范围的数目之间存在着一种线性关系。实施范围检测工作的“准备时间”也几乎会随着范围数目的增大而呈线性增长，不过可以使用流水线技术减少这一时间。在该实验所使用的 Altera Stratix 目标设备上，这一准备时间的变化很大——对于拥有较少范围的策略为一点几个周期；而对于拥有数百个范围的策略则为六个周期。而一次 DFA 转移所需的“循环时间”则几乎不受范围数目的影响。平均循环时间约为 6ns，非常接近于目标设备的最大速度。

FPGA 的运行时钟并不高，200MHz 是一个普通频率。相反，FPGA 是利用计算的并行性来实现高性能的。诸如数字信号处理、面部识别、计算机视觉以及入侵检测等应用都能够充分利用并行性。同时，由于这些应用都是以吞吐量为目的的，因此对于延迟并不太敏感。由于一个频率为 200MHz 的 FPGA 周期为 5ns，因此其引用监视器仅增加不到两个周期的延迟。

设计提示：面积、准备时间、循环时间和吞吐量。准备时间是实施范围检测所需的时间，而循环时间是一次 DFA 转移所用的时间。在策略的复杂度与引用监视器的面积和准备时间之间存在着一种线性关系。策略的复杂度不会影响循环时间，循环时间总是为一个周期左右。FPGA 系统的关键性能指标不是延迟，而是吞吐量。

5.8 使用策略编译器

下面通过将设计流程应用到一项简单的隔离策略中来对其用法加以说明。第一步是创建一个包含下列内容的文件“toy. policy”，如下

```
Isolation;
Compartment1 - > Module1;
Compartment1 - > Range1;
Compartment2 - > Module2;
Compartment2 - > Range2;
```

注意，隔离策略已经使用高级语言进行了表达。

指定范围：在文件“ranges”中指定各个范围，如下所示：

```
0000000 000000f
0000010 000001f
…
```

该文件的第一行指定了 $Range_1$，第二行指定了 $Range_2$，依此类推。文件的每一行都包含了范围的起止地址，通过空格隔开。每个范围必须是对齐的 2 的幂。下一步是针对该策略使用编译器，即

```
%./run. sh toy
not found
0 is a start state
There are 1 unique states
This graph does NOT contain a cycle.
```

脚本“run. sh”实施的步骤如下：

1） run. sh 对范围进行处理。
2） run. sh 由 toy. policy(高级策略规范)创建一个文件 toy. p(低级策略规范)。
3） run. sh 通过句法分析程序来运行 toy. p 。
4） 最后得到的正则表达式作为输入值提供给 RegEx，后者再创建一个文件 grail

_ machine。该文件是一个采用 Grail 格式表达的 DFA。RegEx 是 Thompson 算法的实现，由 Gerzic 提出子集构造[8]，采用一种与 Grail 相兼容的格式修改为输出状态机。

5）run. sh 通过 Grail 来运行 grail _ machine 从而生成文件 gm _ toy。该文件是最小化的 DFA。只需对 Grail 稍加修改即可处理无符号的长整数。

6）run. sh 由 gm _ toy 创建文件 toy. v。该文件是采用 Verilog HDL 语言对执行该策略的引用监视器所作的描述。

7）run. sh 检查 DFA 是否具有隐蔽信道。

8）run. sh 创建文件 toy. dot，该文件采用 Graphviz 格式表达 DFA。

9）run. sh 通过 Graphviz 格式来运行 toy. dot，进而生成文件 toy. ps、该文件是 DFA 的 PostScript 版本并能够打印或在屏幕上显示。图 5-16 显示了该图形。

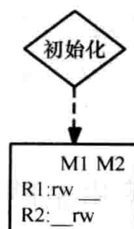


图 5-16　与演示设计流程的 toy 隔离策略对应的 DFA（确定性有限自动机）

文件 toy. p 是低级策略规范，它是由高级策略规范 toy. policy 生成的：

```
Access0 - > {Module1,rw,Range1};
Access1 - > {Module2,rw,Range2};
Policy - > (Access0|Access1) *;
```

文件 toy. v 是描述强制执行 toy. policy 的引用监视器的 Verilog HDL 代码，具体如下：

```
module State _ Machine(clock,
                       reset,
                       module _ id,
                       op,
                        address,
                        is _ legal);
input clock,reset;
input[4:0]module _ id;
input[1:0]op;
input[31:0]address;
output is _ legal;
reg is _ legal;
reg[0:0]state;
parameter s0 =' d0;parameter s1
=' d1;
wire r0;
```

```
wire r1;
assign r0 = (address[31:4] = =28'd1)? 1'b1:1'b0;
assign r1 = (address[31:4] = =28'd2)? 1'b1:1'b0;
always @ (state)
begin
     case(state)
               s0:
             is _ legal = 1'b1;
          s1:
             is _ legal = 1'b0;
          default:
          is _ legal = 1'b0;
     endcase
  end
always @ (posedge clock or posedge reset)
if(reset)state = s0;
else
     case(state)
          s0:
             case({module _ id,op,r0,r1})
             9'b000101101://2 3 1
                       state = s0;
             9'b000011110://1 3 0
                       state = s0;
             default:
                   state = s1;
             endcase
        s1:
        state = s1;
     default:
        state = s1;
   endcase endmodule
```

采用上述 Verilog 代码所表达的引用监视器有三个输入值（module _ id、op 和 address）和一个输出值（is _ legal）。它使用刚好一个周期的时间来根据策略对所请求的存储访问的合法性加以确定。op 有四种可能的值："00" 表示既不读取也不

写入；“01”表示仅读取；“10”表示仅写入；“11”表示既读取也写入。assign 语句对各个范围同时进行检查，以确定 address 的范围。表达式 {module _ id，op，r0，r1} 连接 module _ id、op、r0 和 r1 从而形成一个单独的转移符号。第一个 case 语句根据 state（状态）是允许还是拒绝来确定 is _ legal 的值。第二个 case 语句则根据当前状态和转移字符来确定下一状态。

5.9 从数学角度构建严格的策略

为了确保一项策略的精确性，该策略必须能够允许所有合法行为而拒绝所有非法行为。要检查策略是否反映出设计人员的意图，需要使用一种自动方法。虽然这是一大难题，不过高级规范语言能够协助缓解这一难题。本节将说明如何检查策略中合法行为与非法行为之间的冲突。

5.9.1 交叉乘积法

一项正确的策略所应具备的条件之一就是同一个行为不能既是合法的又是非法的。获取合法访问语言与非法访问语言之间的交集的结果应当是空集。否则（如果有任何的重叠部分）就必须告知设计人员，以便其能够加以修正。要计算两项策略的交集，只需使用 Grail（尤其是 fmcross 命令）来获得其状态机的交叉乘积即可。该计算过程的代价是状态机大小的二次方。图 5-17 中的 Venn 图解展示了这一基本思路。

图 5-18 是一幅 Venn 图解，显示了一种构建策略的增量法。合法访问规范的“草稿”可以通过这样的方式进行检查：判断已知的非法行为具体实例与合法规则之间是否有任何重叠。该自动化程序能够测试大量已知的非法行为并在出现重叠时告知设计人员。

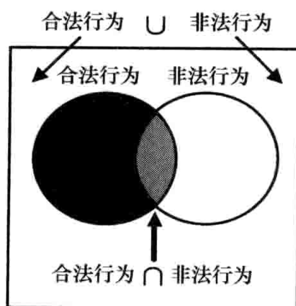


图 5-17　合法行为与非法行为交叉的错误策略的 Venn 图解

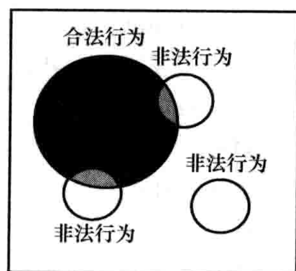


图 5-18　增量式策略构建的自动化方法的 Venn 图解。可以对照合法访问的规范“草稿”对已知的非法行为的几种实例自动进行检查，以确定是否存在任何交叉

5.9.2　实例

思考一下用字母A、B、C、D、E建立的一种合法行为语言的简单实例 L_{Legal} = (A|B|C)*。图5-19展示了允许 L_{Legal} 的DFA。假设同时还有一种非法行为语言 $L_{Illegal}$ = (C|D|E)*。图5-20展示了允许 $L_{Illegal}$ 的DFA。图5-21展示了DFA允许 $L_{Legal} \times L_{Illegal}$，即C*。

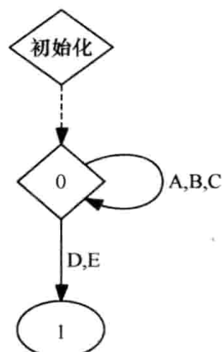


图5-19　识别该语言(A|B|C)*的DFA。输入值为D或E时会导致该DFA转变为拒绝状态(状态“1”)

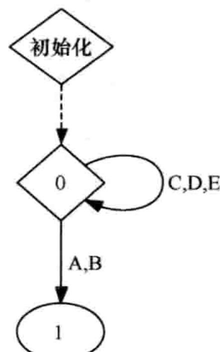


图5-20　识别语言(C|D|E)*的DFA。输入值为D或E时会导致该DFA转变为拒绝状态（状态“1”）

这一方法还可被用于计算B&L和Biba策略的交集。图5-22展示了B&L策略的DFA（确定性有限自动机），所不同的是当前明确显示出了向拒绝状态（状态“1”）的两次额外转变。图5-23展示了Biba策略的DFA，所不同的是当前明确显示出了向拒绝状态（状态“1”）的两次额外转变。图5-24展示了识别B&L与Biba策略的交集的DFA，这是通过获得其各自的DFA的交叉乘积后而计算得出的。

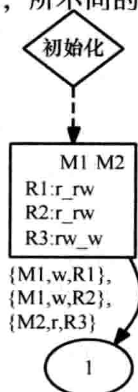


图5-22　这是5.5.4小节中的B&L策略的DFA。输入值为{$Module_1$，w，$Range_1$}或者{$Module_1$，w，$Range_2$}时，可导致该DFA转变为拒绝状态（状态“1”）

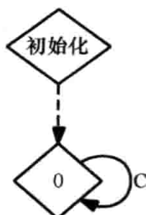


图5-21　识别语言C*的DFA

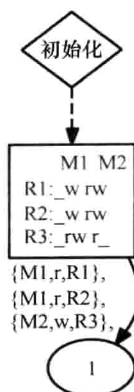


图 5-23 这是 5.5.6 小节中的 BibaL 策略的 DFA。输入值为 {$Module_1$，r，$Range_2$} 或者 {$Module_2$，r，$Range_2$} 时，可导致该 DFA 转变为拒绝状态（状态“1”）

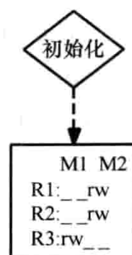


图 5-24 该 DFA 识别出了 B&L 和 Biba 策略的交集，从而使两项策略均得到执行。该交集是通过获得其各自的 DFA 的交叉乘积计算得出的

5.9.3 单一的策略变化

两项策略交集的判定能力对于动态策略也非常有用。在具备动态切换策略能力的系统中，交叉乘积法能够确保策略的所有变化均为单一性变化。例如，如果系统从一项较为宽松的策略切换为一项较为严格的策略，那么某个核有可能在该新策略生效之后仍然在其局部存储器中保留某些敏感信息。尽管对该数据的访问在原来的策略下是合法的，但是新的策略禁止这种访问。一个不成熟的解决方案会在更改策略之后对系统中的所有核进行清理，但这样做的代价会很大，而且可能会中断重要的服务。另一种解决方案则是始终只切换到更为宽松的策略。在一个只允许切换为单一宽松策略的系统中，每项策略都是先前策略的一个超集。换而言之，$Policy_i$ 与 $Policy_{i+1}$ 的交集与 $Policy_i$ 是相同的。假设在某个动态策略系统内存在着一个策略集合 {$Policy_1$，$Policy_2$，$Policy_3$，…，$Policy_N$}。为了确定哪些策略变更是单一性的，设计人员需要获得每一对策略（$Policy_i$ 与 $Policy_j$）的交集，并检查其结果是否与 $Policy_i$ 相同。如果相同，那么从 $Policy_i$ 到 $Policy_j$ 的变更就属于单一的宽松型变更。

5.9.4 混合策略的形式化要素

如图 5-25 所示，一个 FPGA 系统由数个处理器核组成，每个核都拥有各自的局部存储器。该系统还拥有片上全局存储器、片外全局存储器和各个处理元件与存储元件之间的互连线路。引用监视器控制着各个核可以访问哪些全局存储区域，以及访问的方式（读取和写入）。实施管理的安全策略可被视为一份规则表。

策略规则可能会因为系统事件而修改。此外，某些组织安全策略包含由用户动作所触发的状态更改，从而会导致一条或多条规则的更改。规则变更带来的一个问题是，当前策略可能会允许某个内核访问局部存储器中的信息，但是随后的策略则会禁止该访问。由于引用监视器不会控制对局部存储器的访问，因此需要引入一种方法来缓和此类被禁止的访问。一种方法是将有问题核的局部存储器进行归零，作为策略变更的一部分；另一种做法是禁止会导致该问题的此类策略变更。这两种方法都需要对该问题的实质进行明确理解。该问题是从全局属性的角度来阐述的，这些全局属性可能会由于策略的变更而被违反。

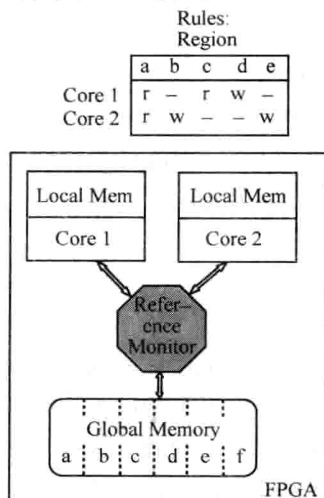


图5-25 FPGA存储控制。一个FPGA系统由数个处理器核组成（本例中为两个），每个核都拥有其局部存储器。系统还拥有全局存储器以及处理元件与存储元件之间的互连线路。引用监视器控制着各个核可以访问那些全局存储区域（a、b、c、d、e、f），以及访问的方式（读取或写入）

策略表P由一个规则集合R = {s，o，a}组成，指明了被允许的访问，其中s是主体（即各个核）集合中的一个元素；o是存储区域集合中的一个元素，而a则是访问模式的枚举型集合中的一个元素，这些模式包括：null、read（读取）、write（写入）、read and write（读取与写入）。全局与局部存储区域内允许进行各种操作。对全局存储器所进行的主要操作抽象表示为g _ mem _ acc（s，o，a）。例如，g _ mem _ aec（s，o，r）会将一个全局存储区域转换为主体（注意：存储器在越是细粒化的水平上越是容易被特征化，比如字节，而非区域）的局部状态（如寄存器）。局部存储器可被表示为local（s）。而读出操作在表示方式上，其某个变量的新值用基本符号（'）来表示，而单个的元素则用下标表示。

$$g_mem_acc(s_1, o_1, r) \Rightarrow local'(s_1) = local(s_1) \cup o_1 \tag{5-1}$$

我们假定，某个读取操作导致数据变成局部存储器的一部分，因为即使其操作主体不将数据直接写入其局部存储器中，它也可以根据数据的值来修改局部存储器（另外，局部存储器也可能被建模以包含本地寄存器）。

我们所关心的全局安全属性是

$$\forall s: subject, o: object, a: (access(g_mem_acc(s,o,a)) \Rightarrow [s,o,a] \in P) \quad (5\text{-}2)$$

即主体仅能够对全局存储器进行被允许的访问。策略变更可以表示为

$$P_change(new: table) \Rightarrow P' = new \quad (5\text{-}3)$$

我们需要确保无论何时 $P' \neq P$，式（5-2）始终保持不变。以下关系将得到满足（与上述两种方法相对应）：

$$P' \neq P \Rightarrow [\ \forall s: subject, o: object,$$
$$a: access([s,o,a] \in P \rightarrow [s,o,a] \in P' | local'(s) = \phi)] \quad (5\text{-}4)$$

即如果策略发生变更，那么原来的策略中的每一种访问都要包含在新的策略中，或者将所有主体的局部存储器都进行清零。

5.10 总结

为了防止不恰当的存储共享和控制存储漏洞，本章描述了指定访问策略等的方法和语言。该语言能够被自动综合为可重构硬件执行模块。策略语言有利于对任意粒度的策略进行表达。为了对这些技术的效率进行评估，通过实验来生成各种不同的策略并使用设计流程来综合硬件模块。这些方法都十分高效，并且在必须加以识别的范围的数量上也具有可扩展性。下一章将展示架构是如何确保执行模块在受保护的同时又能为每一次存储访问所调用的。

参 考 文 献

1. A. Aho, R. Sethi, J. Ullman, *Compilers: Principles, Techniques, and Tools* (Addison Wesley, Reading, 1988)
2. Altera Inc, Quartus II Manual, 2004
3. J.P. Anderson, Computer security technology planning study. Technical Report ESD-TR-73-51, ESD/AFSC, Hanscorn AFB, Bedford, MA, 1972
4. D.E. Bell, L.J. LaPadula, Secure computer systems: mathematical foundations and model. The MITRE Corporation, Bedford, MA, USA, May 1973
5. K.J. Biba, Integrity considerations for secure computer systems. Technical Report ESD-TR-76-372, USAF Electronic Systems Division, Bedford, MA, 1977
6. D.F.C. Brewer, M.J. Nash, The Chinese wall security policy, in *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, 1989
7. D.E. Denning, A lattice model of secure information flow. Commun. ACM **19**(5), 236–243 (1976)
8. A. Gerzic, CodeGuru: write your own regular expression parser, November 2003, http://www.codeguru.com/
9. T. Huffmire, S. Prasad, T. Sherwood, R. Kastner, Policy-driven memory protection for re-

configurable hardware, in *Proceedings of the European Symposium on Research in Computer Security (ESORICS)*, Hamburg, Germany, September 2006

10. T. Huffmire, B. Brotherton, G. Wang, T. Sherwood, R. Kastner, Moats and drawbridges: an isolation primitive for reconfigurable hardware based systems, in *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2007
11. T. Huffmire, T. Sherwood, R. Kastner, T. Levin, Enforcing memory policy specifications in reconfigurable hardware. Comput. Secur. **27**(5–6), 197–215 (2008)
12. S. Johnson, Yacc: yet another compiler-compiler. Technical Report CSTR-32, Bell Laboratories, Murray Hill, NJ, 1975
13. M. Lesk, E. Schmidt, Lex: a lexical analyzer generator. Technical Report 39, Bell Laboratories, Murray Hill, NJ, October 1975
14. P. Linz, *An Introduction to Formal Languages and Automata* (Jones and Bartlett, Sudbury, 2001)
15. J. Navarro, S. Iyer, P. Druschel, A. Cox, Practical, transparent operating system support for Superpages, in *Fifth Symposium on Operating Systems Design and Implementation (OSDI'02)*, Boston, MA, December 2002
16. D. Raymond, D. Wood, Grail: A C++ library for automata and expressions. J. Symb. Comput. **11**, 341–350 (1995)
17. J. Rushby, A trusted computing base for embedded systems, in *Proceedings 7th DoD/NBS Computer Security Conference*, September 1984, pp. 294–311
18. J. Saltzer, Protection and the control of information sharing in Multics. Commun. ACM **17**(7), 388–402 (1974)
19. F.B. Schneider, Enforceable security policies. ACM Trans. Inform. Syst. Secur. **3**(1), 30–50 (2000)
20. G.W. Smith, R.B. Newton, A taxonomy of organisational security policies, in *Proceedings of the 23rd National Information Systems Security Conference*, Baltimore, MD, USA, October 2000
21. D.F. Sterne, On the buzzword "security policy", in *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, Oakland, CA, 1991, pp. 219–230
22. E. Witchel, J. Cates, K. Asanovic, Mondrian memory protection, in *Tenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, San Jose, CA, October 2002
23. M.E. Zurko, R.T. Simon, User-centered security, in *Proceedings of the 1996 Workshop on New Security Paradigms*, Lake Arrowhead, CA, September 1996

第6章　采用壕沟技术的空间隔离

摘要： 本章描述了壕沟与吊桥技术（Huffmire 等人，《Proceedings of the 2007 IEEE Symposium on Security and Privacy》，2007 年 5 月于美国加利福尼亚洲奥克兰市），即一种在可重构式单芯片上隔离多个核的方法。壕沟技术通过一种可验证的方式将多个核放置在芯片的不同区域来实现逻辑上的隔离。吊桥技术采用互连跟踪技术来对以下两个方面进行静态验证：第一，各系统元件之间仅允许合法连接；第二，传送敏感数据的接口未被窃听或被错误地连接到其他核或 I/O 上。为促进各个核之间的合法通信，对两种可供选择的通信架构进行了比较。

6.1　概述

考虑一种由两个处理器核和一个可共享的 AES 加密核组成的 FPGA 系统，所有元件都部署在同一个 FPGA 芯片上。有关该系统的进一步描述可参考本书第 7 章。三个核都需要对片外存储器进行访问，以便存储和获取数据。如何才能确保某个处理器的加密密钥不会被其他处理器通过外部存储器或直接从加密核中读取密钥而将其窃取呢？这些系统没有虚拟存储器，而由 CAD 工具所生成的电路中逻辑门与导线交织而成，复杂难解。要防止密钥从加密核中被窃取，需要在逻辑门层次上将加密引擎与其他核隔离开来。要在外部存储器中对密钥加以保护，需要执行一个本书第 5 章中所述的存储保护模块。此外，还必须确保每一次存储访问都要经过引用监视器，并且该引用监视器也要加以隔离和保护。同时还必须确保各个核只可通过指定的接口进行通信，以防止对加密核中的密钥进行越权访问，同时防止互连线路上的非法获取。对设计流程的最后某一阶段稍加修改即可对系统元件的布置加以限制。本章所展示的技术着眼于构建一个极具凝聚力的可重构式系统设计方法，该方法支持在一块芯片上使用不同信任级别的核来构筑系统。

6.2　隔离

隔离是有关计算机安全的一个基本概念，它结合了隔离和受控共享的理念。对密码系统（如加密设备等）而言，最先需要研发强有力的隔离手段，因为通过红线传送的分类明文必须与通过黑线传送的密文隔离开来。Saltzer 和 Schroeder 将完整的隔离定义为一种“将主体分隔为彼此之间无任何信息流或彼此不受控制的数

个隔离区的保护系统”[7]。如果系统的各个部分完全被隔离，功能性将被大大降低，因此，需要一种技术来促进各个隔离区之间数据的受控共享。在 FPGA 系统中实现受控共享的一种解决方案是采用壕沟技术实现隔离，采用吊桥技术提供精确共享的手段。

6.3　采用壕沟技术的物理隔离

由于综合工具以性能为目标对设计布局进行优化，所以生成电路的逻辑元件和互连线路往往盘根错节，如图 6-1 所示。每个方块代表一个单独的开关盒以及相关的查找表和布线。对于一个拥有 30K 开关盒的 FPGA，对比特流进行静态分析在计算上并不可行。为了保护核的数据并防止对核的运行造成干扰，以下章节将对壕沟的用法加以说明，以确保对核进行隔离。

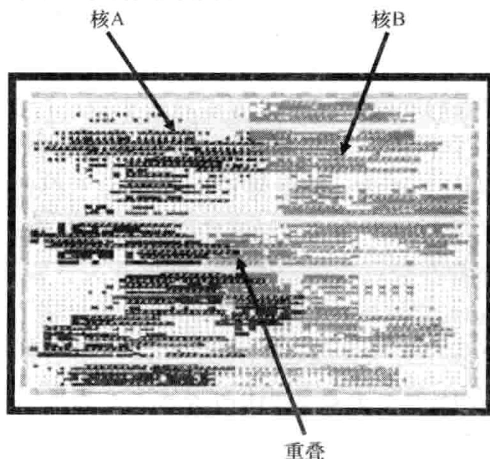


图 6-1　映射到小型 FPGA 上的一个简单的双核系统（每个灰色或黑色方块代表一个具有线路高度复杂性的开关盒。为了防止核的重叠，必须对设计工具施加约束，因为核的重叠会增大产生非预期信息流的风险。为了使大型设计的静态分析任务具有可运算性，这些约束也是必须的）

6.4　构建壕沟

通过对设计工具稍加约束，壕沟就能够从空间上对各个核进行隔离，以增强安全性。设计工具如 Xilinx 公司的 PlanAhead 软件等能够帮助设计人员对核布局进行精细的控制，从而为设计过程提供极大便利。本节将介绍两种构建壕沟的方法，这两种方法都是将核安置在不同的物理区域内。

1）间隔法需要用盲区（即一条壕沟）将每个核围住。如果在设计中仅使用长

度小于壕沟尺寸的线路片段，则壕沟内部区域将完全被隔离。

2）检查法会对靠近核边界的线路片段进行系统的检查，以确保无任何片段进入核内部，从而能够减少或消除盲区，并放宽对线路片段使用上的限制。靠近核边界的片段要比远离边界的片段受到的限制更大。

在这两种方法中，核都只能通过一条称为“吊桥”的精确定义的路径来与外界进行通信。间隔法采用的是空间上的隔离，而检查法采用的是逻辑上的隔离。

6.4.1 间隔法

间隔法使用一条物理“壕沟”将每个核包围，在壕沟区域内的开关盒都已被关闭，只留有一条精确定义的路径（即“吊桥”，稍后加以探讨）进行核间通信。Xilinx 公司的 Virtex 系列 FPGA 采用可跨越 1、2 或 6 个可配置逻辑块（CLBs）的线路片段。长线资源跨越一整行或一整列。由于每个开关盒都引入了延迟，因此绕过中间的开关盒可以极大地提高性能。间隔法为了实现空间上的隔离，牺牲了通过长线获得的性能。

图 6-2 展示了构建壕沟的间隔技术。为了对该实例进行扩展，如果将设计工具约束为只使用长度为一个和两个单位的布线片段，则壕沟的尺寸至少要为两个单位，以防止信号越过壕沟。通常，如果将设计限制为不得使用长度超过 w 的布线片段，那么壕沟的宽度需要至少为 w。对这些属性的静态验证很方便。如果一个开关盒内的所有布线晶体管都被设置为未连接，则该开关盒属于一条壕沟。同时还必须确认所有与长度超过 w 的片段相连的开关都已关闭。进行这一确认需要一些有关比特流的信息，包括每个开关配置位的位置。由于 JBits API 能够提供这一信息，因此这些方法适用于 JBits API 所支持的任何架构[3]。另外，将 JBits 扩展到其他架构对于希望这样做的厂商来说也很方便。

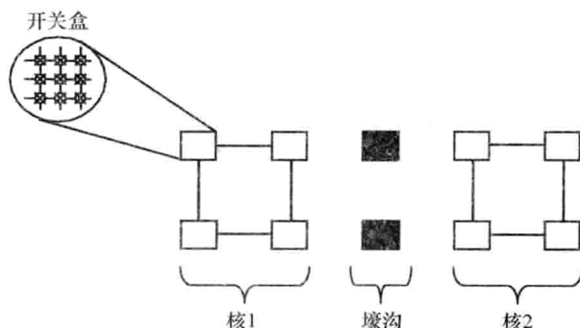


图 6-2　间隔法用盲区将每个核围住。本实例约束了其构建过程只能使用可以跨越一个开关盒的布线片段。受该约束条件的影响，壕沟的宽度必须在一个单位以上。由于以黑色表示的开关盒是被禁用的，因此从核 1 到核 2 无法使用一个单位长度的连线。但是两个单位长度的线路片段可以越过壕沟连接另一个核

6.4.2　检查法

使用系统性检查能够减少甚至不使用壕沟。“无缝”壕沟是一种没有任何间隔的壕沟（即不存在任何关闭的开关盒）。这一重要理念就是允许使用长度超过 w 的片段，前提是只要它们距离边界足够远。例如，即使 w 为“2”，在一个 20×20 见方的核的中央也能使用一个十六进制的片段。显然，使用静态分析来确定片段没有越过边界非常重要，但是在一个核中其实只须检查一小部分连接线路。从核边界到中央所需的校验会逐渐减少，从而使得设计工作更加方便，如图 6-3 所示。例如，对于一条无缝壕沟，至少在距离边界一个可配置逻辑块长度的区域内不必校验长度为一个单位的片段。在距离边界两个可配置逻辑块的区域内不需检查长度为两个单位的片段。由于长线会跨越整行或整列，因此始终需要对其进行校验。

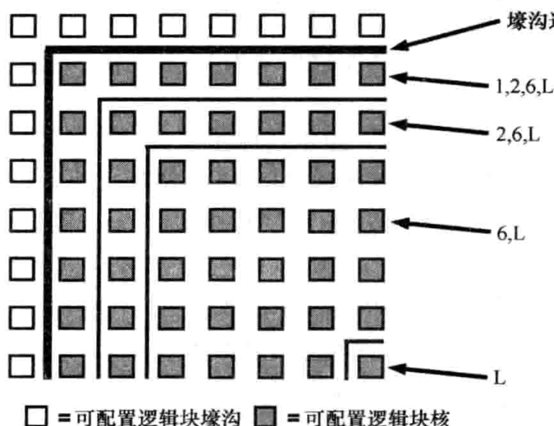


图 6-3　检查法在壕沟构建过程中所产生的间隔更小或根本没有间隔，这种壕沟被称为“无缝”壕沟。设计工具必须对越过边界的片段进行静态校验。从核边界开始直到中心位置，所需的校验会逐渐减少

在可配置逻辑块中，必须进行查找的片段深度 d 取决于片段的长度 l（1、2 或 6）以及壕沟的宽度 w，即

$$d(l,w)=\begin{cases}0(\text{当 } l<w \text{ 时})\\ l-w(\text{当 } l\geqslant w \text{ 时})\end{cases}$$

该方程表明，如果 w 为 1，则不必检查核中长度为 1 的片段，因为该类片段无法越过壕沟。即使是在边界上的片段也必须利用壕沟内的其他连接线路来避开，这一情况会在验证过程中被检测到。

6.4.3　间隔法与检查法的比较

对于间隔法，通过对电路性能与壕沟面积之间的权衡分析可以发现，壕沟越

小，电路性能越差[5]。

若限制设计只使用短线会迫使连接线路穿过更多的开关盒，每个开关盒都会产生一定程度的延迟。同时也需要更多的开关盒资源和更大的电路面积。第一项实验规定几条基准电路只能使用长度为 1 个单位的片段。第二项实验则将这一情形与规定电路只使用长度为 1 和 2 片段的结果相比较。第三项实验则规定电路只能使用长度为 1、2 和 6（即没有长线）的片段。然后将这三项实验的结果与不加限制的“基准条件”（可使用所有长度的片段）进行比较。在使用 VPR 进行电路布局布线后，对面积和关键路径的性能进行了测量。虽然取消长线收效甚微，但取消十六进制线路会严重影响到面积和性能，而且取消长度为 2 的片段还会进一步加剧情况的恶化。为了更好地理解有用的逻辑、膨胀和无效区的关系以及它们对面积的影响，引入“有效利用率”进行度量。有效利用率为自由逻辑与受限逻辑和壕沟面积之和的比率。假定各个核为统一的矩形，则尺寸为 2 的壕沟的有效利用率是最高的，除非核的数量非常大（ >100）或者非常小（为 1）。如果仅有 1 个核，则不必使用壕沟。

通过对检查法的分析可以发现，较小的壕沟比较大的壕沟拥有更为出色的性能，原因是对于使用较长的线路片段的限制条件被放宽了。一项实验将检查法应用到了三个系统上，包括一个双核处理器片上系统[6]、一个多输入多输出收发模块和一个 JPEG 编码器。在壕沟尺寸分别为 6、2、1、0 的情况下与不使用壕沟的基准条件下，对系统的面积和性能的比较。不出所料，壕沟较小的系统使用的面积要小得多，但是需要更多的校验。静态校验带来的额外开销只是一小笔的一次性代价。使用检查法来构建的壕沟，其性能受到的影响是最小的。虽然无缝壕沟使用的面积最小，但是使用尺寸不为零的壕沟通常更为有利，原因是壕沟区域能够作为吊桥布线的通信信道使用。下一节中将对吊桥技术进行探讨，吊桥为核与 I/O 引脚之间的信号而严格定义的路径。

6.5 使用吊桥的安全互连

虽然壕沟能够对核进行隔离，但核之间必须能够以受控方式进行通信。“吊桥”为各个核之间或核与 I/O 之间的通信提供了一条精确定义的路径。各个核的位置以及所允许的连接线路必须预先指定。吊桥技术适用于多种互连架构，包括直连线路和共享总线。未来的工作将会把吊桥技术向片上网络的方向扩展[2]。

6.5.1 直连的吊桥技术

系统设计人员必须首先指定合法的连接线路，以便对设计是否符合规范进行静态分析。参考文献［5］中所述的路线跟踪工具会对比特流和描述模块与互连线路

的文件进行操作，并且它不需要有关核的设计细节（如HDL），因为这些有可能属于专利技术而难以获取。在设计流程结束时进行校验有助于发现在设计早期引入的非法连接。

壕沟允许对每个核的位置和有效的连接进行精确的规范。该规范类似一个文本文件，定义了所有核和I/O引脚，包括它们的位置和一份合法连接的列表。互连跟踪涉及比特流的分析，以便确定开关盒的状态，从而对某条连接线路的布线路径进行跟踪。这降低了设计当中出现非法连接的风险，通过跟踪程序会检测到这些非法连接。

路线跟踪工具会需要两个输入值，即一个比特流文件和上述规范文件。连接是按照来源（引脚或模块）和目的地（引脚或模块）来指定的。跟踪算法是从一份输入及输出引脚（某些引脚可能既可用于输入也可用于输出）的列表开始的，这些引脚能够接入或接出可配置逻辑块。在对所有输入引脚进行跟踪后，接下来会跟踪从模块内的所有可配置逻辑块引出的所有对外连接。最后，对从模块引出的所有对外连接进行反向跟踪。如果设计正确无误，那么最后一个步骤将不会发现任何连接，因为之前的各个步骤应该都已经发现了。路线跟踪算法是一种简单的横向优先搜索，并做了几点修改。它会保留一份已访问过的引脚列表，以防止将同一路径重复搜索。一旦到达下一模块，搜索就会终止。跟踪程序会输出它所发现的所有连接，并且能够选择性地列出显示某个连接整个路径的路线树。当全部完成后，跟踪算法提示设计是否已成功得到验证。以下伪代码描述了跟踪过程：

```
RouteTree trace(pin,module){
    add pin to routeTree
    for all sinks of wire this pin is on {
      if sink is connected to pin
        if sink has already been searched
          return
        if sink is in another module
        check if connection is valid
        return
      add sink to list of searched pins
      trace(sink, module)
  }
}
```

路线跟踪工具的输入文件的格式。路线跟踪工具输入文件的首行应以字母"D"开头,并指定设备类型,如:

D XC2V6000 FF1517

该文件的下一行以字母“N”开头并指定模块、引脚和连接的数量，如：

N 4 5 12

该文件的下一行以字母“M”开头并指定模块的名称及其坐标在 X 轴上的最小值、最大值和 Y 轴上的最小值、最大值，即

M MB1 11 35 57 80

M MB2 11 35 13 35

M MB3 54 78 57 80

M MB4 54 78 13 35

该文件的下一行以字母“P”开头并指定引脚的名称以及引脚为输入端、输出端还是复位端，如

P B25 rst #Reset

P C36 in #rs _ 232 _ rx _ pin

P J30 out #rs _ 232 _ tx _ pin

P C8 in #rs _ 232 _ rx2 _ pin

P C9 out #rs _ 232 _ tx2 _ pin

该文件的下一行以字母“C”开头并指定连接：连接源、目标地和宽度，如

C B25 MB1 1

C C36 MB1 1

C MB1 J30 1

C B25 MB2 1

C MB1 MB2 32

C MB2 MB1 32

C B25 MB3 1

C MB3 C9 1

C C8 MB3 1

C B25 MB4 1

C MB4 MB3 32

C MB3 MB4 32

图 6-4 显示了该路线跟踪过程。图中黑线显示的是连接两个设计部件的路线。路线跟踪工具从该路线位于图中左上角的起点开始直到位于图中右下角的终点进行跟踪。

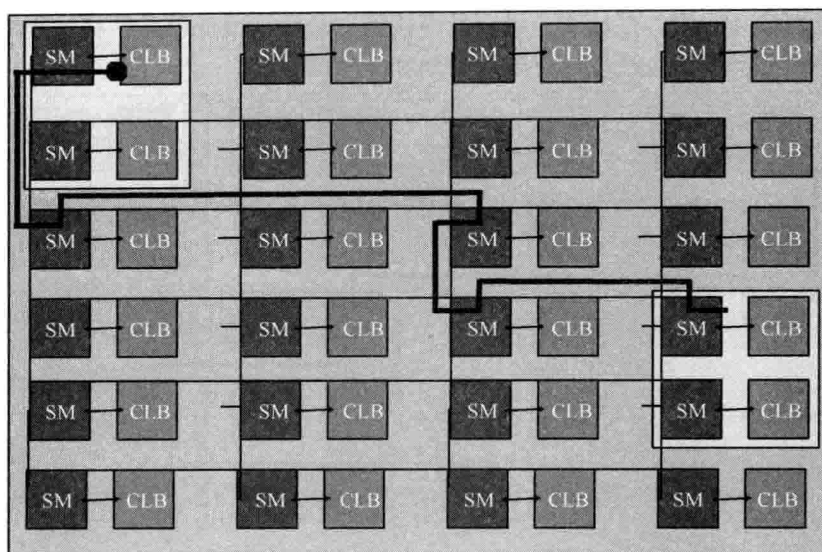


图6-4　路线跟踪。图中黑线显示的是从位于左上角的浅灰色区域开始一直延伸到右下角的浅灰色区域的一条路线，区域中包含两个可配置逻辑块和两个开关矩阵。该路线跟踪工具始终遵循着这一路线，从起点直到终点

6.5.2　局部重构的路线跟踪

图6-5显示的是局部重构的路线跟踪过程。局部重构会使路线跟踪过程更为高效，因为只需要为穿过可重构部分设计的每个连接存储两个引脚即可。对于接入但未从该区域接出的连接线路，只需存储一个引脚及其连接方向（接入或接出）。只有接入或接出可重构部分设计的连接才需要被搜索。该效率增益的代价是初始跟踪的少量额外开销。图6-6显示的是采用了局部重构的设计的FPGA Editor的视图。

6.5.3　共享总线架构的吊桥技术

除了直连线路外，核还可以通过一条共享总线进行通信。但是，由于其他的核可能会窥探总线流量，因此总线的共享特性会带来安全问题。即使总线流量被加密，总线也可能被用来对时标信道实施隐蔽攻击，即由"高安全级别"核向"低安全级别"核发送数据。高安全级别核会对其总线参考值进行调整，而低安全级别核则会察觉到该调整。要处理这些问题，可使用仲裁器来对总线实施时分多路复用。每个核只有在为其分配的时间片内才能对总线进行访问。虽然将时间按核数量均等划分会降低总线性能，但是可以消除总线上的隐蔽时标信道。

为缓解时分多路复用造成的性能下降，Hu提出了允许低安全级别核向高安全级别核"贡献"时间的思路[4]。循环过程从安全级别最低的核开始，并循序进行

到安全级别最高的核。在从安全级别最高核到安全级别最低核的转移过程中，需要进行某些内部操作，包括耗尽贡献的时间和清除缓存等。

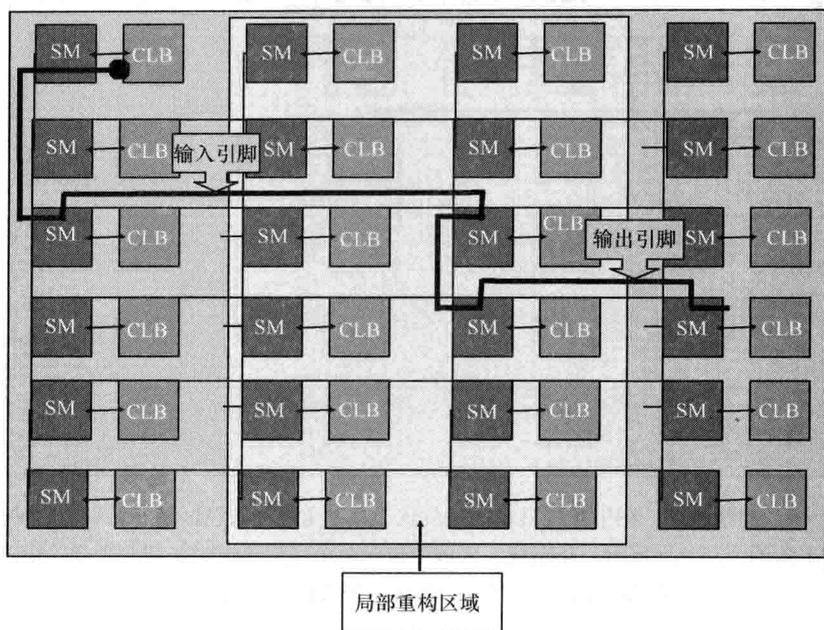


图 6-5　局部重构的路线跟踪。只有接入或接出可重构部分设计（用浅灰色表示的中间区域）的引脚才需要被搜索，从而使得跟踪过程快很多

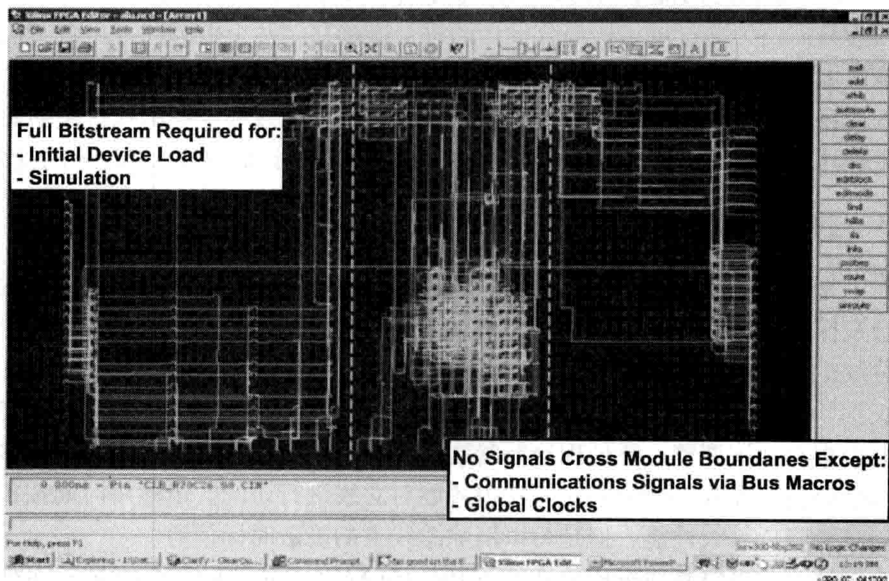


图 6-6　采用局部重构设计的 FPGA Editor 视图

“有序循环”方案比贡献方案具有更大的灵活性，同时还能提供更高的保密性，并争取比其固定时间片更多地使用 CPU 的机会。在这种方法中，会为进程的各个等价类分配一段很小的固定时间片。根据等价类标记的顺序将其放置在循环调度队列中，以便低级别的进程能够在高级别进程之前执行，不过从最高级别向最低级别的转移除外。图 6-7 显示了一个循环。每个类最多可以用完为其分配的时间片，也可以在这之前放弃处理器的控制权。

此外，在该“有序循环”方案中，在最低类之后每个等价类会依次获得由较低类所贡献的富裕时间[1]的机会。在安排最高的等价类后，它需要用完本次循环中全部所剩余的贡献时间。因此，最低类不会察觉到较高类所使用的 CPU 时间数量上的任何变化，而较高类也仅能察觉到由较低类所导致的变化。

为了确保最高类能够获得捐献的时间（如有的话），可以对中间类加以管制，例如将其使用的捐献时间限制在某个指定的百分比之内。如果该百分比随着级别的上升而递增，那么结果类似于为较高级别的访问类提供了更高的优先级。

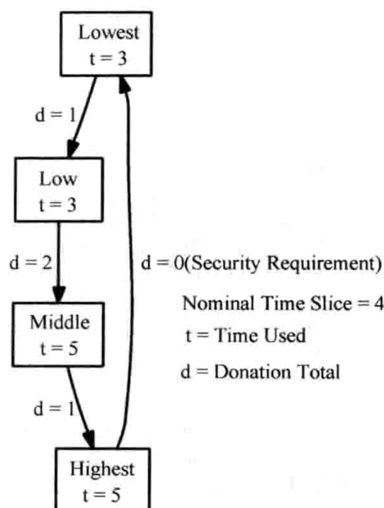


图 6-7　有序循环调度

用于保障 MLS 调度的“固定”循环法会在循环调度方案中为每个等价类提供一段固定的时间片。但是，如果一个进程根本就不需要为其分配的时间片，这一做法就有可能浪费资源。当全部 MLS 标记被排序时，可以采用“有序优先级”调度方案，时间片的分配顺序与 MLS 标记成反比。在需要为保密性较高的进程提供较高优先级的系统中，该方案会产生问题。此外，如果有一个或多个未分级标记元件与某个分级标记元件组合在一起时（即存在标记的部分排序时），必须为处于该层次级的所有进程分配相同的优先级（或在有序循环方案中分配相同的时间片），即组成等价类。

除了隐蔽信道外，还必须防止非法窥探。在每个核与总线之间放置一个仲裁器，可以将每个核对总线的使用限制在为其分配的时间片内。总线与存储器的连接需要使用本书第 5 章中所述的引用监视器技术。每个核都可以拥有一个仲裁器（这需要一个中央定时多路复用器来处理调度），或者采用一个仲裁器来供所有核使用。参考文献［5］表明，采用一个仲裁器实际上更高效，并且可以将该仲裁器整合到片上外围总线（OPB）中[6]。

6.6 采用壕沟技术来保护引用监视器

本书第 5 章所述的引用监视器必须加以隔离、不可旁路，并可以检验。要保护引用监视器不受破坏，可以采用壕沟将其隔离。要防止引用监视器被旁路，可以采用吊桥来防止核直接对存储器进行访问，对存储总线进行窥探，或是与另一核建立非法连接。互连跟踪技术可以确保存储器 I/O 控制块只与引用监视器相连接。

参考文献

1. A. Bavier, L. Peterson, D. Mosberger, BERT: a scheduler for best effort and realtime tasks. Princeton University Technical Report TR-602-99, Princeton, NJ, March 1999
2. S. Bourduas, Modeling, evaluation, and implementation of ring-based interconnects for network-on-chip. Ph.D. Dissertation, McGill University, Dept. of Electrical and Computer Engineering, Montreal, Canada, May 2008
3. S. Guccione, D. Levi, P. Sundararajan, JBits: Java-based interface for reconfigurable computing, in *Proceedings of the Second Annual Conference on Military and Aerospace Applications of Programmable Logic Devices and Technologies (MAPLD)*, Laurel, MD, USA
4. W.M. Hu, Lattice scheduling and covert channels, in *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, Oakland, CA, May 1992
5. T. Huffmire, B. Brotherton, G. Wang, T. Sherwood, R. Kastner, Moats and drawbridges: an isolation primitive for reconfigurable hardware based systems, in *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2007
6. T. Huffmire, B. Brotherton, N. Callegari, J. Valamehr, J. White, R. Kastner, T. Sherwood, Designing secure systems on reconfigurable hardware. ACM Trans. Des. Automat. Electron. Syst. (TODAES) **13**(3), 44 (2008)
7. J. Saltzer, M. Schroeder, The protection of information in computer systems. Proc. IEEE **63**(9), 1278–1308 (1975)

第 7 章　综合运用：设计实例

摘要： 本章通过一个设计实例将前面各章中所述的安全原型综合在一起。实例中的嵌入式系统与两个需要加密的独立网络相连接。该系统包含两个处理器内核和一个共享的 AES 加密内核，上述内核均安装在同一个 FPGA 芯片上。类似系统的更多细节可以在 Huffmire 等人的《ACM Transact. Des. Automat. Electron. Syst.》（TODAES）（13（3）：44，2008）中找到。

嵌入式系统设计实例对于理解前面各章阐述的安全技术如何应用于具体设计至关重要。系统设计实例还可以展示如何制定现实的安全策略。该系统设计实例还有助于分析安全技术的适用性及其对系统性能和复杂度的影响。本章专门对在多核可重构嵌入式系统中如何应用第 5 章中介绍的引用监视器技术、第 6 章中介绍的壕沟与吊桥技术，以及第 3 章中介绍的隐蔽信道分析技术，做了详细的介绍。

7.1　多核可重构嵌入式系统

图 7-1 展示了该设计实例。这是一个连接到两个独立网络的嵌入式系统，每个网络都在特定的级别上做了标记，在该级别上，两个网络上的数据都必须进行加密。该系统包含两个 MicroBlaze 软 CPU 核、一个共享的 AES 加密核、以及两个以太网接口控制器，所有的部件都在一个 FPGA 设备上运行。将多个部件整合到一个设备上可以节约能耗、成本和面积。该系统还包含了片外 DRAM。这些部件被安排在了两个隔离区内，灰色隔离区包含其中一个 CPU 核和一个以太网接口，黑色隔离区包含另一个 CPU 核和另一个以太网接口。每个隔离区都需要与标有各自安全级别的网络进行通信。隔离区通过片上外围总线相互连接，该总线包含仲裁逻辑和一个引用监视器。

图 7-1 中的灰色隔离区与黑色隔离区共用一个加密模块以避免重复配置（因空间的限制）。除 AES 引擎外，两个隔离区不得共享其他任何资源。在一个隔离区正在使用 AES 模块时，另一个绝对不允许对其进行访问，否则可能会导致隔离区之间出现信息泄漏；AES 模块在被不同域使用前后必须进行清除。实际上，AES 模块的级别会因其使用者的不同而变化，即可能是临时性单级设备，也可能是“周期处理”设备[6]。周期处理是一个专门的术语，指的是在每项机密工作进行之后和被下一个使用者使用之前对某个共享资源进行“清洁处理”。清洁处理通过将先

前工作所残留的所有存储内容全部清除来完成。

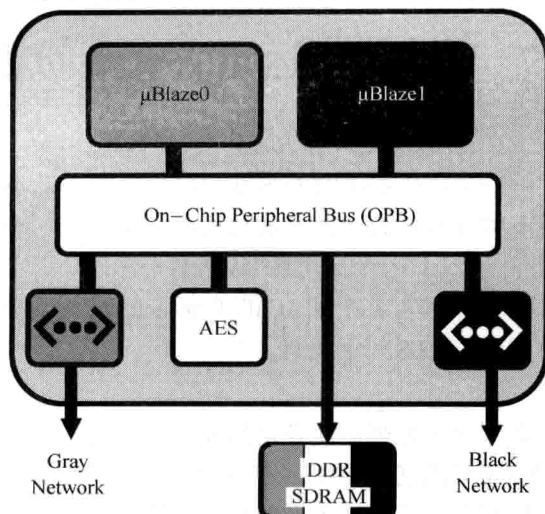


图 7-1　由两个处理器内核、一个共享的 AES 加密内核、两个以太网接口和一个共享的外部存储器组成的系统，所有元件都通过一条共享总线相连。该片上总线外围中集成了一个引用监视器和仲裁逻辑

为了符合这些设计要求，壕沟会在空间上对该可重构设计中的所有核进行隔离，同时引用监视器还会执行时间隔离策略，以便两个隔离区能够对 AES 模块进行“周期处理”，确保两个隔离区的相互分离。在每个策略周期之后，AES 引擎会对数据进行清除。

7.2　片上外围总线

由于传统共享总线的工作是将多个组件连接在一起，而不是将其隔离，因此必须对总线进行修改，从而对模块间的共享资源进行控制。片上外围总线是 Xilinx 的嵌入式开发包（EDK）中提供中的一个 IP 软核，用于连接嵌入式系统中的各个外围设备。通过对 OPB 总线进行适当修改可以集成引用监视器，该监视器能够执行将灰色隔离区与黑色隔离区从逻辑上加以分隔的策略。两个 CPU 核是主设备，而加密核、以太网接口，以及 DRAM 为从设备。

7.3　AES 核

我们可以采用共享存储器的方式，使用定制的控制器对 AES 加密核进行控制。

需要加密或解密的处理器会将需加密或解密的明文放入处理器隔离区中的共享DDR SDRAM缓冲区内。然后，处理器会通过向特定的控制字写入数据来向AES核发送信号。该信号的内容包括究竟是加密还是解密、数据的位置以及数据的大小。在完成转换后，AES核会马上将密文或明文写入到共享存储器缓冲区内，并通过控制字向处理器发送信号。最后，发起转换的处理器会从共享缓冲区中复制被转换的数据。

由于AES核实际上属于共享设备，因此对它的访问必须由引用监视器来控制。每个缓冲区都位于发起转换的处理器的存储区域内。这可以防止各个处理器相互读取对方的明文。此外，引用监视器还会执行一项状态性的“周期处理”策略[1,5]，该策略可以确保每一次只有一个处理器能够使用AES核。

7.4 逻辑隔离区

图7-1展示的是该嵌入式系统的灰色和黑色隔离区。灰色隔离区包含了其中的一个MicroBlaze处理器和一个以太网接口。黑色隔离区包含了另一个CPU核和以太网接口。两个隔离区对AES核进行“周期处理”，而DRAM则被划分给了这两个隔离区和AES核。由于灰色和黑色网络在不同级别上运行，因此两个隔离区之间保持分离是十分必要的。为了实现这一分离，壕沟技术提供了对各个核的空间隔离，而引用监视器则实现了各个系统部件在时间上的分离。

7.5 引用监视器

在该嵌入式系统中，与总线相集成的引用监视器会将访问传达给外围部件、片上存储器以及片外存储器。引用监视器可以同意或拒绝对存储器的访问请求，同时还能对连接片上外围总线（OPB）的任何部件的访问进行仲裁。仲裁机制确保了两个核不能够同时使用总线。

存储器映射I/O技术使得采用引用监视器能够访问I/O设备和共享存储器，同时还使得按比例扩充主设备的数量成为可能。

7.6 状态性策略

状态性策略的形式化顶级规范是使用第5章中所述的低级策略语言来表达的。$Module_1$ 对应于 $\mu Blaze_0$，$Module_2$ 对应于 $\mu Blaze_1$。策略编译程序会将该规范直接转换为执行该策略的引用监视器的硬件描述。引用监视器能够执行资源共享策略，因为系统中除MicroBlaze处理器之外的每个部件都被分配了一个特定的地址范围，用

以下语句来表达：

$Range_1 \rightarrow [0x28000010, 0x28000777]; (AES_1)$

$Range_2 \rightarrow [0x28000800, 0x28000fff]; (AES_2)$

$Range_3 \rightarrow [0x24000000, 0x24777777]; (DRAM_1)$

$Range_4 \rightarrow [0x24800000, 0x24ffffff]; (DRAM_2)$

$Range_5 \rightarrow [0x40600000, 0x4060ffff]; (Ethernet_1)$

$Range_6 \rightarrow [0x40c00000, 0x40c0ffff]; (Ethernet_2)$

$Range_7 \rightarrow [0x28000004, 0x28000007]; (Ctrl_Word_1)$

$Range_8 \rightarrow [0x28000008, 0x2800000f]; (Ctrl_Word_2)$

$Range_9 \rightarrow [0x28000000, 0x28000003]; (Ctrl_Word_{AES})$

AES_1 是灰色隔离区中的共享 DDR SDRAM 缓冲区，而 AES_2 则是黑色隔离区中的共享 DDR SDRAM 缓冲区。第一种策略状态（$Access_0$）对应于 $Module_1$ 和 $Module_2$ 均未使用 AES 核的情形，这是系统的初始状态。第二种状态（$Access_1$）则对应于 $Module_1$ 正在使用 AES 核时的情形。而第三种状态（$Access_2$）则对应于 $Module_2$ 正在使用 AES 核时的情形。处理器通过向 Ctrl _ Word1（$Range_7$）写入控制字可获得对 AES 核的访问权，该写入动作会触发一次状态转换（对于 $Module_1$ 为 $Trigger_1$；对于 $Module_2$ 则为 $Trigger_3$）。处理器通过向 $Ctrl_Word_2$（$Range_8$）写入控制字来释放 AES 核，该写入动作又会触发一次状态转换（对于 $Module_1$ 为 $Trigger_2$；对于 $Module_2$ 则为 $Trigger_4$）。下列语句指定了三种状态：$Access_0$、$Access_1$、和 $Access_2$。

$Access_0 \rightarrow \{Module_1, rw, Range_5\} | \{Module_2, rw, Range_6\}$

$| \{Module_1, rw, Range_3\} | \{Module_2, rw, Range_4\}$

$Access_1 \rightarrow Access_0 | \{Module_1, rw, Range_1\} | \{Module_1, rw, Range_9\};$

$Access_2 \rightarrow Access_0 | \{Module_2, rw, Range_2\} | \{Module_2, rw, Range_9\};$

下列语句指定了 $Trigger_1$、$Trigger_2$、$Trigger_3$ 和 $Trigger_4$ 各种状态之间的转换：

$Trigger_1 \rightarrow \{Module_1, w, Range_7\};$

$Trigger_2 \rightarrow \{Module_1, w, Range_8\};$

$Trigger_3 \rightarrow \{Module_2, w, Range_7\};$

$Trigger_4 \rightarrow \{Module_2, w, Range_8\};$

下列语句使用了一些相当复杂的正则表达式来指定执行策略的状态机的结构：

$Expr_1 \rightarrow Access0 | Trigger_3 Access_2 * Trigger_4;$

$Expr_2 \rightarrow Access1 | Trigger_2 Expr_1 * Trigger_1;$

$Expr_3 \rightarrow Expr_1 * Trigger_1 Expr_2 *;$

$Policy \rightarrow Expr_1 * | Expr_1 * Trigger_3 Access_2 *$

$| Expr_3 Trigger_2 Expr_1 * Trigger_3 Access_2 *$

$|\mathrm{Expr}_3\mathrm{Trigger}_2\mathrm{Expr}_1 * |\mathrm{Expr}_3|\varepsilon$;

这些策略的制定暴露出了低级语言的局限性。在此情形下，鉴于正则表达式的复杂性，使用 Grail 语言[4]来指定状态机并生成正则表达式则更为方便，它能使用“fmtore”（有限状态机到正则表达式）程序将状态机转换为正则表达式。第一步是在名为“grail _ machine”的文件中使用 Grail 语言来指定 DFA 的基本结构：

```
(START)|-0
0 A 0
1 B 1
2 C 2
0 D 1
1 E 0
0 F 2
2 G 0
0 -|(FINAL)
1 -|(FINAL)
2 -|(FINAL)
```

图 7-2 展示了该 DFA（确定性有限自动机）。接下来，“fmtore”程序会根据有限状态机来生成正则表达式。

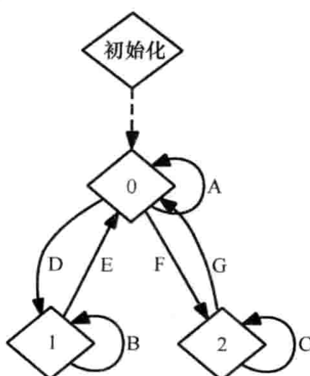


图 7-2　在此情形下，需要借助于 Grail 来生成正则表达式

```
%./fmtoregrail _ machine
(A+FC*G)*
+(A+FC*G)*FC*
+((A+FC*G)*D(B+E(A+FC*G)*D)*)E(A+FC*G)*FC*
+((A+FC*G)*D(B+E(A+FC*G)*D)*)E(A+FC*G)*
+((A+FC*G)*D(B+E(A+FC*G)*D)*)
```

+""

注意，A 对应于 $Access_0$，B 对应于 $Access_1$，C 对应于 $Access_2$，D 对应于 $Trigger_1$，E 对应于 $Trigger_2$，F 对应于 $Trigger_3$，G 对应于 $Trigger_4$。

该问题的解决方案包括对第 5 章所述的高级语言进行扩展，从而处理更加多样化的状态性策略，或者为工程师提供一种拥有用户界面的策略构建工具，然后自动生成策略。

图 7-3 展示了状态性策略的 DFA。该设计流程根据策略自动生成了对引用监视器的硬件描述。在系统的初始状态下，两个核都不会使用 AES 核。这一状态即为 init 所指向的状态，而 $Access_0$ 的访问矩阵也显示了出来。$Module_1$ 通过向 control _ $word_1$ 写入控制字来获得对 AES 核的访问权，导致向左下角的状态进行转换，这里显示的是 $Access_1$ 的访问矩阵。由于 $Module_1$ 能够访问 control _ wordAES 和 AES_1，因此它能够将明文或密文复制到 AES 的存储器中相应的位置（AES_1）上，并通过 control _ wordAES 向 AES 核发送相应的信号。在完成加密或解密后，AES 核通过 control _ wordAES 对 $Module_1$ 发送信号，$Module_1$ 从共享缓冲区中复制被转换的数据。然后，$Module_1$ 通过向 control _ $word_2$ 写入来释放 AES 核，从而触发一次回到初始状态的转换。$Module_2$ 也以类似的方式来获得和释放对 AES 核的访问权。引用监视器可防止 $Module_1$ 和 $Module_2$ 同时使用 AES 核。该方案将策略的各条规则编码

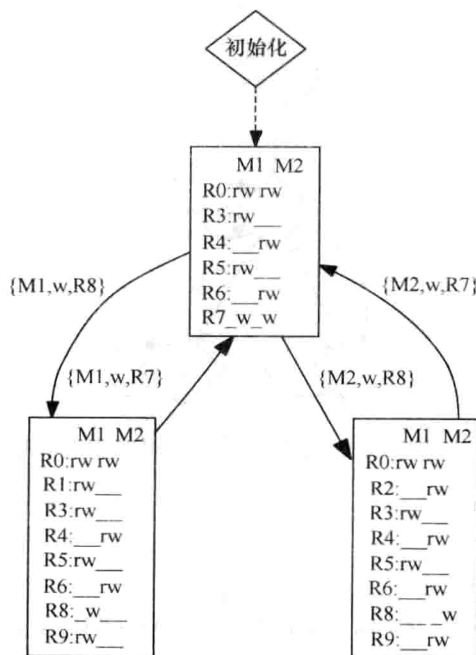


图 7-3　执行嵌入式系统状态性策略的 DFA（确定性有限自动机）

到硬件当中，使得硬件按照设计来执行该策略。

7.7　安全的互连可扩展性

为了高效安全地管理核间通信，拥有大量内核的系统需要采用比拥有单个引用监视器的单条总线更加复杂的策略。根据互连的不同选择，可能需要用到分布式引用监视器。对于某些状态性策略，多个引用监视器的状态同步化可能会是一项挑战。一个安全的系统架构能够通过减少所需引用监视器的数量来将同步化的开销降到最低限度。例如，可以将多个核组织成空间上相隔离的等价类，一个组内的各个核只能通过本地互连线路来相互通信。引用监视器只需对越过等价类边界的通信进行监控。

未来的嵌入式片上系统将使用高级互连技术，包括片上网络栅格[2]。在这些设计中进行安全管理需要将执行机制集成到片上路由器内。

7.8　隐蔽信道

在图 7-3 所示的实例中，使用第 3 章所述的隐蔽信道分析技术可以对从 $Module_1$ 到 $Module_2$（从灰色处理器到黑色处理器）各种可能的隐蔽信道进行分析。换而言之，灰色处理器可能会把引用监视器的内部状态作为一条隐蔽的存储通道向黑色处理器发送信息。可能的隐蔽信道有几条。一条基于访问总线的干涉。另一条基于灰色处理器在策略 $Access_1$ 和 $Access_0$ 之间调整系统的能力，以及黑色处理器检测到这些变化的能力。该隐蔽信道的带宽取决于灰色处理器抢占 AES 核的频繁程度。其中有一种解决方案是时分多路访问（TDMA）的共享机制。也就是说，灰色隔离区可以在一段固定时间内使用 AES 核，然后再由黑色隔离区使用，依此类推。该系统还可以要求 AES 核被使用足够长的一段时间，从而使得隐蔽信道的带宽足够低。该系统还可以测量灰色核占用 AES 核的次数，并在越过某一阈值后报警。该系统还能够通过随机改变系统事件来将噪声引入隐蔽信道。

为了避免 AES 核本身的内部状态被利用来从灰色核发送信息到黑色核的问题，需要使用对象重用机制。一种十分严厉的方法是使用动态局部重构来擦除 AES 核所在的壕沟边界区域的所有内容，然后将全新的 AES 核配置信息重新载入该区域。一种更为巧妙的对象重用技术则是仅擦除 AES 核的状态化元素，不过这一更为精细的方法需要缜密的分析。你怎么知道擦除了所有内容而没有留下任何东西呢？

7.9 壕沟技术与吊桥技术的合并

将壕沟技术与吊桥技术合并到设计当中可以实现空间上的分离，并简化验证工作。壕沟技术对包括引用监视器本身在内的系统部件在空间上进行隔离，从而对引用监视器加以保护。正确配置吊桥技术可以检测到系统部件之间的任何非法连接，从而防止引用监视器被旁路。尤为重要的是，跟踪算法可以检测到从各个核和存储器中引出的绕过引用监视器的连接。它还能检测到各个核之间绕过引用监视器的非法连接，以及允许各个核窥探彼此的存储器流通量的非法连接。

Xilinx 公司的 PlanAhead[7] 可以用来构建壕沟。如图 7-4 和图 7-5 所示，PlanAhead 提供了一个可视化接口用于将核放置在芯片上，同时设计人员可以在各个核之间留出空隙。PlanAhead 的输出结果是一个在综合步骤中要用到的用户约束文件。该综合工具会在每个核的有限区域内计算出最佳布局。

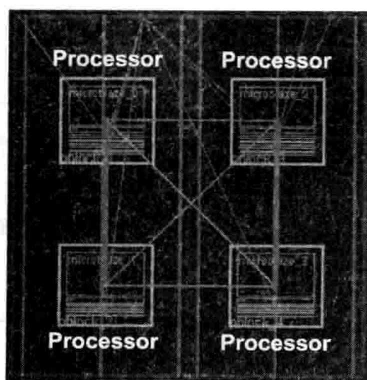


图 7-4　使用 PlanAhead 工具为四个 MicroBlaze 处理器核设计构建壕沟

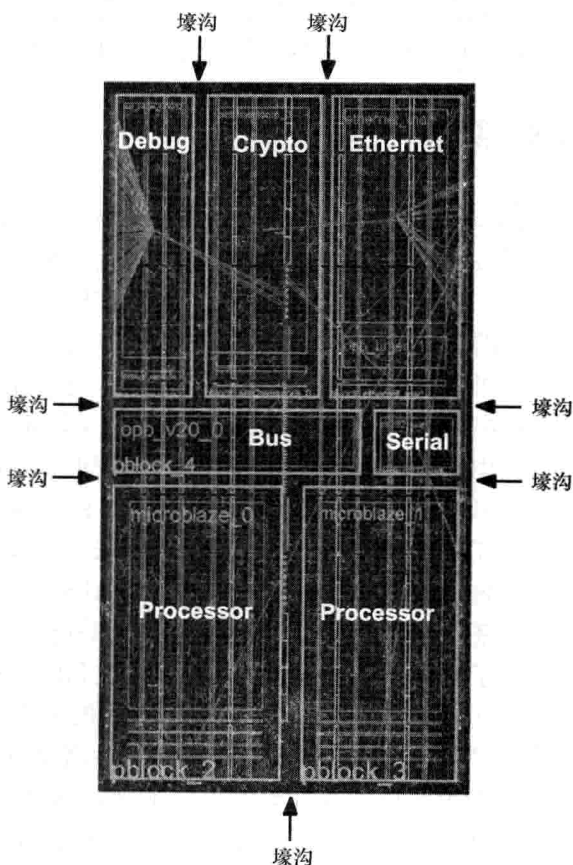


图 7-5　被划分为七个由壕沟加以防护的核后在 PlanAhead 工具中观察到的系统布局图

设计划分为七个不同的区域，每个区域对应一个部件。确定各个核的最佳布局对系统性能有着极大影响，并需要反复试验。需要相互进行通信的核应当放在一起，而其他核也应当紧靠 I/O 引脚放置。我们可以使用多次布局布线技术对不同的结果进行比较。

7.10 实施与评估

Xilinx Platform Studio(XPS)软件被用来集成嵌入式设计，而 Modelsim 则使用一组测试激励对核和定制的 OPB 总线(片上外设总线)进行测试。策略编译程序生成对引用监视器的 Verilog 描述。XPS 还被用来开发运行在 FPGA 之上的软件来驱动 MicroBlaze 处理器。Xilinx Microprocessor Debugger(XMD)用于对软件进行调试。通过实验发现，构建无缝壕沟的方法对性能的影响微乎其微，而且对面积也没有任何影响[3]。

7.11 软件界面

继续以前面的灰色/黑色隔离区为例，编程接口允许运行在 PC 或笔记本电脑上的应用程序以程序控制的方式向设备发送数据和密钥，并接收返回的密文或明文。通过 C + + 软件来实现的用户界面，可以指定操作指令（加密或解密）、输入的数据文件、密钥文件、以及保存结果的位置。在典型配置中，一台“灰色”笔记本电脑通过其中一个以太网接口连接到 FPGA 板卡，而另一台“黑色”笔记本电脑则通过另一个以太网接口连接到 FPGA 板卡。

7.12 安全可用性

安全技术只有在设计人员能够方便地对其加以充分利用时才会起到作用。构建该嵌入式系统的经验表明，壕沟技术是一种简单而有效的部件平面布置方案，并且实施起来直接明了。采用最佳性能来确定该部件平面布置方案需要进行反复试验，但这对于有经验的设计人员来说同样是简单而明了的。Xilinx 公司提供的 OPB 架构能够很便捷地集成引用监视器。策略编译器同样有利于引用监视器的集成，因为修改策略只需重新编译即可。经验还表明，使用类似本书所介绍的定制工具来构建策略是必须的，因为这并不要求设计人员必须是一位正则表达式方面的专家。

7.13 更多的安全架构示例

在一个符合 Bell 和 LaPadula 信息流策略的系统中，计算资源是局部排序的。

例如，信息能够从“C”核流向“S”或者“TS”核，但不会从“C”核流向“U”核。如果存在很多核和很多直接连接，那么在实施用规则表描述的策略中，规则条目的数量就会变大，从而使引用监视器的规模变的很大。为每个直接连接而重复生成多个引用监视器也会造成资源的浪费。可以在直连线路上放置一种类似于“二极管”的装置，只允许信息向一个方向流动，从而避免为每个直连线路配置一个引用监视器。在壕沟隔离的隔离区之间使用二极管可以极大地减少传达通信所需的仲裁/引用监视器资源。这种简易方法是以减小执行粒度为代价的。

7.13.1 设计的种类

FPGA 系统安全架构的设计空间可以分为两类，即一体化架构以及内核—存储器的架构。

1. 一体化架构

该类设计只考虑整体的计算资源而不考虑存储器。图 7-6 显示了该类设计的一个实例。

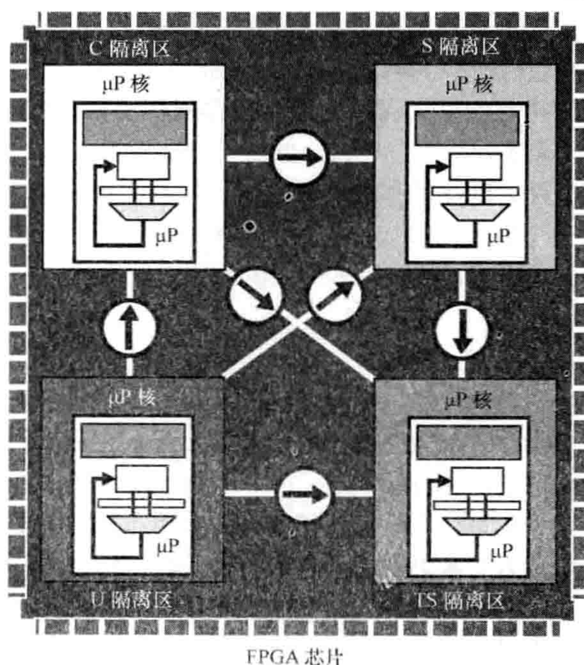


图 7-6　该系统拥有四个使用壕沟隔离的安全隔离区。每个隔离区包含一个 CPU 核。这四个隔离区呈“网状”结构排列，而“二极管”则可以确保隔离区之间信息的正确流向。数据能够从 U 隔离区流向 C 隔离区，从 U 流向 S，从 U 流向 TS，从 C 流向 S，从 C 流向 TS，以及从 S 流向 TS。该类设计未将存储器考虑在内

2. “内核-存储器”架构

该类设计将存储器考虑在内，并且把存储器划分为几个区域。给每个区域分配一个安全标记，同时也为各个核指定同样一套标记，从而建立“隔离区”或“分区”。由于存储器能够驻留在芯片上或芯片外，因此该类设计还能够进一步划分为片上存储器、片外存储器，或两者兼顾的安全架构。片上可重配置的引用监视器可以为片上 SRAM 和片外 DRAM 提供存储保护。壕沟可以为片上 BlockRAM（BRAM）提供保护。图 7-7 展示了该类设计的一个实例。

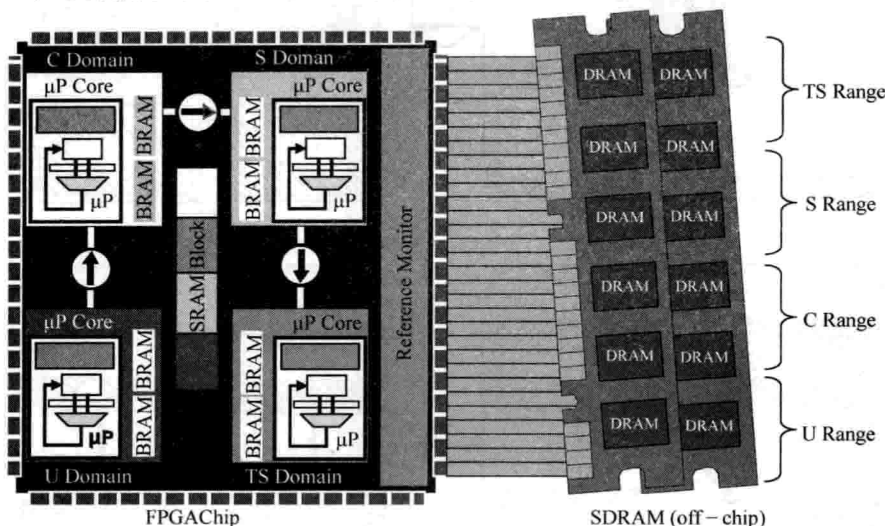


图 7-7 该系统代表了同时兼顾片上和片外存储器的一类安全架构。每个隔离区包含一个 CPU 核和两个采用壕沟隔离的 BlockRAM（BRAM）区块。引用监视器为片上 SRAM 和片外 DRAM 提供存储保护

引用监视器所执行的策略用于指定哪些核可以访问哪些存储区。DRAM 被划分为若干区域，并且给每个区域分配一个安全标记。各个核与上述区域使用的是同一套标记。为了符合 Bell 与 LaPadula 信息流策略，引用监视器绝对不能允许一个核写入安全性比自己低的区域或者读取级别更高的区域。

7.13.2 拓扑结构

“拓扑结构”是核的不同排列方式。图 7-8 显示了按层次方式排列的各种通信拓扑结构。以一种恰当的方式将计算资源划分为若干隔离区，能够极大地降低策略执行机制的复杂性和面积。通过对引用监视器进行合理布局和使用二极管也能够降低既定策略执行机制的复杂性。由于在硅产品生产领域取得的进步，一块芯片上可以集成越来越多的核。由于核之间的可能的直连线路的数量随着核数目的增长是二

次方关系，因此，以一种高效而安全的方式对所有这些通信进行管理就变得非常重要。

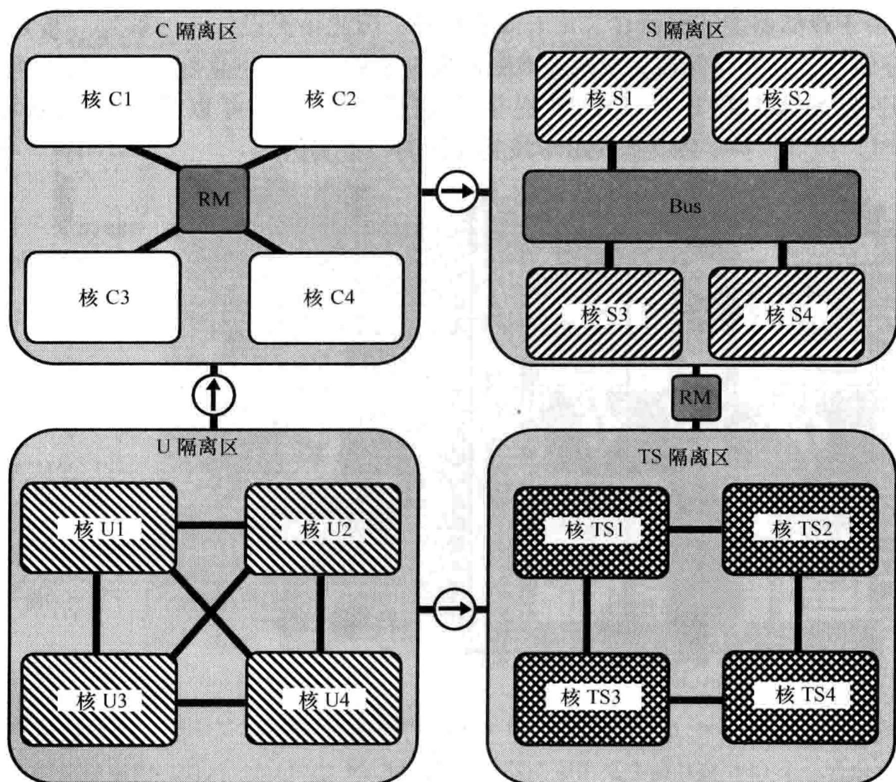


图 7-8　该图显示的是按层次方式排列的各种通信拓扑结构。TS 隔离区所拥有的四个核是呈网格状排列的。S 隔离区拥有的四个核是通过一条总线相连的。C 隔离区所拥有的四个核是呈“星形”结构排列的，并带有一个拥有最低权限的引用监视器。U 隔离区所拥有的四个核是呈网状排列的。二极管可以确保数据只从 U 流向 TS，从 U 流向 C，以及从 C 流向 S，TS 与 S 之间的连接线路上放置了一只引用监视器，以便执行更为精细的策略（比如仅在某些情况下从 S 流向 TS）

二极管对于某些直接连接而言是一种理想的执行工具，因为可以非常方便地在硬件中建立定向连接，而且它比引用监视器要小得多。唯一需要的是找到一种方法来检验二极管是否依照策略被放置在了需要它们的位置上，以及它们是否指向了正确的方向。但是，某些情况下引用监视器比二极管更为可取，比如在状态性策略当中，必须临时性禁止某个特定的通信链路的情况。

7.14 总结

必须把安全性作为嵌入式系统设计时首先考虑的最重要约束，而嵌入式系统经常被错误地认为是安全的。上面的设计实例展示了本书阐述的安全要素是如何应用的。构建上述系统的经验还表明，自定义工具对于构建涉及复杂正则表达式的状态性策略很有帮助。以后可以开展的工作还有许多，其中包括将引用监视器以一种高效的方式集成到直接存储访问（DMA）控制器内，并为使用DMA控制器的系统设计相应的策略等。拒绝服务的问题还有待解决，尤其是不断发送非法请求的恶意内核的问题。

参考文献

1. J.P. Anderson, Computer security technology planning study. Technical Report ESD-TR-73-51, ESD/AFSC, Hanscom AFB, Bedford, MA, 1972
2. S. Bourduas, Modeling, evaluation, and implementation of ring-based interconnects for network-on-chip. Ph.D. dissertation, McGill University, Dept. of Electrical and Computer Engineering, Montreal, Canada, May 2008
3. T. Huffmire, B. Brotherton, N. Callegari, J. Valamehr, J. White, R. Kastner, T. Sherwood, Designing secure systems on reconfigurable hardware. ACM Transact. Des. Automat. Electron. Syst. (TODAES) **13**(3), 44 (2008)
4. D. Raymond, D. Wood, Grail: A C++ library for automata and expressions. J. Symb. Comput. **11**, 341–350 (1995)
5. US Department of Defense, National Industrial Security Program Operating Manual (NISPOM), 28 February 2006
6. C. Weissman, Secure computer operation with virtual machine partitioning, in *Proceedings of the National Computer Conference and Exposition*, Anaheim, CA, May 1975
7. Xilinx Inc., PlanAhead Methodology Guide, San Jose, CA, 2006

第 8 章　前瞻性问题

摘要：本章考虑了若干前瞻性问题，包括可靠的工具、硬件设计的形式化验证、配置管理、编程语言、物理攻击、设计的失窃、以及整个制造供应链的安全性。

8.1　可信的工具

创建能完全消除隐蔽信道的硬件设计仍然是一大难题。为了开发出切实可信的设计流程，今后必须继续研究，这一研究需要付出长期的巨大努力。例如，FPGA 设计工具极为庞大和复杂，包含优化功能等诸多特性。各种工具都有所有权问题，而且各销售商还在不断增加工具的新功能。安装完整的 FPGA 设计工具可能要耗用数十 GB 的硬盘存储空间。但是，使用基本的功能子集来完成较为小巧的设计流程也是有可能的。这类设计流程可以由小规模的设计团队来开发，并对其进行严格的形式化安全分析，从而比完整版本的工具提供更高水平的可信性。如果将综合优化功能移除，那么性能方面可能只要求部分设计能够进行可信的编译（如安全机制）。这类只具备最基本功能的设计工具虽然有局限性，但并不意味着所生成的与安全性密切相关的硬件模块就一定比采用完整的商业化套装工具所生成的效率更低。

开发出精简而又切实可靠的设计流程并不容易。在软件行业，Thompson 在其著名论文中就描述过寻找编译程序的精简版本是多么艰难[10]。此外，在一般情况下，通过分析 IP 核或者计算机程序来确定它是否恶意是行不通的，因为，根据 Rice 的定理，这种分析相当于停机问题[7]。

可信性的另一个重要方面是，硬件设计流程的每个阶段的输出结果都应该忠实地执行当前阶段的输入参数。应当对现有编译程序验证技术的应用进行调查[3]。设计当中的某些模块的彻底物理隔离有助于形式化验证，因为那些当前未被测试的模块可以在分析时将其“遮盖”。假定每个被隔离部件的正确行为都被明确定义，那么就可以对每个部件单独进行测试，以确保设计的输出结果正确无误。如果各个模块在运行期间相互影响，则通常必须对其既定的行为进行测试。除了静态分析之外，还应该开发运行机制来检验电路是否正确运行，并验证电路完整性。

8.2 安全系统的形式验证

如何能够将广为熟知的软件安全技术高效地转换到硬件领域[5]需要进行广泛的研究。将形式化方法应用到软件设计过程中涉及使用定理证明和模型校验等现有技术，以确保软件符合安全性要求。特别要对照形式化顶层规范（FTLS）对代码进行检查。不幸的是，定理证明和模型校验并非灵丹妙药，因为在模型、校验、规范的实施过程或者物理部署环境中都有可能存在缺陷。此外，由于安全性分析的复杂性会随着接受分析代码数量的增多而增大，因此可扩展性对模型校验而言也可能会是一个大问题。尽管在这一点上定理证明要比模型校验具有更强的可扩展性，但是前者是一场持久战，并且需要经验丰富的人员开发各种模型并对定理论证加以引导。自动定理证明是一个 NP 完全问题，因此我们希望进一步使用在模型校验中已经用到的各种技术，以便缩小极其庞大而又可以分割的搜索空间。

使用系统评估的通用标准有许多现实的局限性。我们通常认为，通用标准仅仅参照保护方案对评估目标进行评估，但是如果该保护方案存在缺陷，这种评估就不一定有用。现实的情况是，各种系统都是以一种反复迭代的方式开发的，形式化方法和通用标准对此都很难适应。当最终产品与项目起步时的系统大相径庭时，对规范所作的修改可能会显著影响采用形式化方法的分析工作和采用通用标准的评估工作。要减少此类问题还有待研究。

如果缺乏构建可信任系统的工具和理论，设计人员在任何设计中都应当特别注意与安全问题密切相关的的各种部件和接口。在有着诸多要求的复杂系统中，没有哪一种安全技术能够独自提供充分的安全性。多种补充性机制必须协同工作，并需要对整个系统进行全盘考虑。这些机制包括运行期间执行过程中所需的统一安全架构、静态分析、形式化方法、密码技术、安全可用性、用户培训以及其他种种机制。虽然在现实世界里不存在天衣无缝的安全性，但是系统必须要设计得尽可能完善，从而迫使对手必须费尽心机才能对单个机器实施攻击，而不是让多个系统如同多米诺骨牌那样，击中一点，全盘即溃[6]。

8.3 安全可用性

要实现出色的安全性就必须要考虑到用户，其中包括最终用户、管理员和开发人员。开发人员在将各种安全技术整合到实际设计当中时，必须能够方便地对其加以使用。最终用户必须接受有关系统的正确使用、配置、管理和更新的培训，并且了解社会工程攻击的风险。必须为用户创建和实施完善的策略，包括入侵检测和审查政策，以便应对知情人员的攻击。安全分析应当考虑到与人员和环境有关的各种

假设。开发用的计算机和工具应当加以管理和保护，部件出厂时应当采用可靠的交付方式。策略内容必须清楚直观地表达，以便开发人员能够据此进行正确的设计。

8.4 硬件可信性

可重构引用监视器技术是将严格的设计和分析应用于 FPGA 设计安全的第一步，还有更多工作需要去做。未来研究的另一个方向是开发一种可靠的虚拟存储器，为 FPGA 提供资源仲裁。同时，为了确保 FPGA 制造商不会通过内部开发人员的攻击在 FPGA 结构中安插恶意电路，还必须进行认真仔细的调查。虽然攻击者不了解哪一项设计最终会被加载到设备中，但是对于高度可信的应用而言，哪怕是可能性微乎其微的此类攻击也是一个严重的隐患。在理想情况下，攻击者试图在设计结构中安插缺陷成功的机会不可能比发生随机事件（即发生随机故障）的概率更大。应当对现有处理器设计验证技术[4]在 FPGA 结构中的应用进行全面深入的调研。

8.5 语言

采用与硬件设计和硬件设计流程正式定义的属性有关的形式化方法将会是另一个非常有用的长期研究课题。例如，形式化方法可能被应用于 Verilog 或 VHDL 等硬件描述语言（HDL）的代码中[2,8]。形式化方法还可能被用于检验 HDL 编译程序的输出结果是否忠实的执行了输入参数。将高级规范转换为低级实现的过程仍然是一项有待研究的挑战。

另一项未来的课题是增强 HDL 语言的开发功能，从而使设计人员更方便地改进设计的安全性。例如，硬件描述语言能够整合与信息流有关的“安全”概念，就像为专用程序设计语言所做的那样。比如，在硬件描述语言中能够指定某条布线路径的安全标记。这些语言的增强功能有可能与分析 HDL 是否与安全属性一致的静态分析技术结合使用，或者与使用了增强功能语言的运行时机制结合使用。此外，配置管理技术也可能被应用于 HDL 代码，就像已经应用于软件代码那样。

在第 5 章中，我们介绍了用于表达形式化顶层规范（FTLS）的自定义语言的使用。FTLS 可以描述指定 FPGA 上各核之间合法共享存储的策略。编译程序可以将该策略直接转换为执行该策略的电路的硬件描述。采用形式化方法可以确保该电路的 HDL 描述的正确性，或者确保该电路忠实地执行了该 HDL 描述，这些工作将留待以后继续研究。由于能够被加载到 FPGA 上的嵌入式系统的设计空间是无限的，而且一次检验只适用于一个系统，因此该类项目的应用前景非常广阔。该作用域还必须考虑构建检验时所处的系统抽象性级别（如芯片级别、板卡级别、系统

级别等)，以及该检验是否被应用于 FPGA 结构本身、比特流加密机制、可重构设计的比特流或是运行在可重构硬件之上的软件等。此外，该策略语言还能够增加其他特性，使得 FLTS 不会产生缺陷，从而杜绝 3. 5 节所述的隐蔽存储信道的产生。

8.6　配置管理

设计人员该将源自软件行业的配置管理技术应用到硬件行业中。例如，通过软件配置管理（CM）建立一个口碑不错的特定版本的工具库（如编译器）。库中的特定版本工具必须要在设计行业中赢得声誉。在安装该工具之前，工程师可以检验下载的压缩文件或磁盘镜像的加密校验和。此外，还应当用已知的测试案例对该设计工具的输出结果进行分析。最后，就像软件配置管理能够被应用于特定版本的源码开放式软件库（如 GNU C Library）那样，硬件配置管理也应当被应用于特定版本的源码开放式核（如 opencores. org 网站上提供的以太网核）。

8.7　供应链的安全防护

除了受信任的设计工具包、工具流程和设计库外，对整个半导体制造业的供应链加以保护也需要可靠的包装、装配和交付工作。为降低恶意硬件“病毒”的危害，需要通过调查研究来了解和分析恶意硬件并对其分类，以便能够将其检测出来。例如，我们要如何才能防止由“恶意核”造成的拒绝服务类攻击呢？正确使用渗透测试和其他形式的动态分析，对于检测特定的缺陷非常有用，尽管它不可能检测到所有的破坏。虽然开发人员进行了功能测试，不过最有效的还是开发人员和外来测试人员都应当进行的渗透试验。对于由谁来做测试以及进行测试的程度等，需要制定具体的标准。两者都应当根据系统的重要程度来确定。了解软件与硬件的渗透测试之间的差别也是必需的。对于恶意软件的研究结果（包括对于在软件中检测后门的理论局限性的发现等[9]也同样适用于硬件。

8.8　针对 FPGA 的物理攻击

要防止针对 FPGA 的物理攻击，就需要对 FPGA 的具体模型进行分析，以确定容易受到物理攻击的弱点，包括探查、功耗分析，热感应通道（使用温度变化来对信息进行编码)，电磁辐射分析、时序分析、以及“打磨—扫描”攻击。该分析结果可以用于将缓解措施（如屏蔽技术）整合到 FPGA 自身或加载到 FPGA 的设计中，或者两者兼有（如篡改感应网和环氧树脂密封材料)。侧信道攻击不仅会危及到由 FPGA 正在处理的密钥和数据，而且还会危及到用来加密比特流的密钥，从而

有可能对因销售商和产品型号而异的比特流加密方案造成破坏。根据信息理论，传统的屏蔽技术存在着理论上的局限性，因此要完全消除侧信道仍然是一个公开的难题。降低这些通道的带宽是目前的技术所能实现的最佳方案。系统设计人员应当在风险分析过程中对这一实际情况加以考虑。

自行配置后伪装成无线电发射机的 FPGA 恶意核是一种值得进一步调查研究的特殊攻击。它还有可能在可重构硬件上设立一根原始的天线，接收器有可能是另一可重构核、电路板上的另一块芯片或是同一房间里的某个其他设备。除了可重构无线电和小型广播设备外，发射机和接收机都有可能被晶片工厂心怀恶意的内部人员添加到芯片的晶元中。可重构恶意核随后可能会连接到采用硬连接的无线电设备。研究人员已经考虑到了将 FPGA 的温度作为隐蔽信道媒介的问题[1]。

8.9 设计盗窃与故障分析

虽然业界已经在比特流加密和身份验证的研究与开发上投入巨资，但是这些机制的力量还不足以抵挡那些处心积虑、资金雄厚、以政府为后盾的对手们的攻击。通过渗透或收买 FPGA 制造商员工就可以轻易地绕过当前的安全防护技术。比特流加密与身份验证机制需要大量相同类型的安全设计方法（安全架构、密钥管理、弱点分析、形式化验证、防篡改、侧信道预防等）来对它们将要保护的设计进行防护。虽然 PUF 是在生成唯一密钥的方法中非常有前景，但是还需要做更多工作才能以一种可靠的方式来生成拥有足够信息熵的 PUF，并在 FPGA 上来生成这些 PUF。

为了在 FPGA 上建立高度可信的系统，必须对设计中要使用的特定 FPGA 的失效模式（如确实存在的话）进行综合分析。例如，在对给定的故障做出响应的过程中，系统可能仅仅只是采取以下操作：中止、安全地失效、继续运行、以功能缺失模式或维护模式运行或者恢复。因此，必须对所有状态和转换过程进行分析。比如，系统是否有可能从维护模式转换到中止状态。再者，还必须了解更大规模的系统是如何得知个别部件发生故障的。设计中要使用的特定 FPGA 的选择需要仔细考虑，因为诸如 IP 保护机制等安全特性会因销售商（Xilinx、Altera、Actel）和 FPGA 类型（Antifuse、Flash、SRAM）以及 FPGA 的具体型号（Virtex4、Virtex5、Virtex6、StratixII、StratixIII 等等）的不同而有所差异。

8.10 局部重构与动态安全

需要通过进一步的研究来将动态安全研究的成果应用到采用局部重构的 FPGA 系统中去。例如，一个系统是否允许策略发生变更、哪一主体会变更策略、策略变

更的频率、策略是事先确定的还是在运行期生成的、是否有可能恢复到先前的策略、系统是否始终向更为严格的策略转换，除核之外的安全机制是否能被重新配置，以及系统是否为热插拔等，这些问题都应当围绕其对局部重构的安全性和性能的影响来加以考虑。同时还需要通过研究允许局部重构在无需禁用加密的情况下运转，并对 ICAP 接口加以保护。

8.11 结论

可重构硬件在关键性系统中的广泛使用迫使 FPGA 行业采用优先考虑安全性的设计原则。盲目地信任硬件会忽略这样一个事实，即硬件也能够实施恶意的攻击。设计一个基于可重构硬件的嵌入式系统和各种软件设计类似，其中也包括复杂的工具流程的运用和源代码的重复使用等。设计工具和 IP 核中的缺陷可以被人利用来对系统实施攻击。

对 FPGA 的攻击各式各样，任何一种都有可能。这些攻击的目的包括窃取知识产权、窃取机密数据和密钥、随意修改硬件以及拒绝服务等。在晶片工厂，心怀恶意的员工可能会添加一些功能或者窃取设计。幸运的是，FPGA 降低了该问题的风险，因为 FPGA 芯片的制造是在安全措施完备的工厂中完成的，而设计被加载到芯片中是在芯片制造完成之后。为了防止在 FPGA 编程之后设计遭窃，制造商集成了比特流解密机制，以便在设备断电后设计仍能以加密的形式存储起来。尽管为开发比特流保护机制已付出巨大的努力，处心积虑的攻击者们还是试图对解密机制展开攻击以便绕开它们进而攫取密钥。数十年来致力于建立防篡改硬件的经验表明，防止一个处心积虑的对手对设备实施物理攻击非常困难，且代价高昂。除了物理攻击之外，对手们还可能试图绕过为防止非授权方远程更新 FPGA 配置而设置的身份验证机制。

为了最大限度地降低存在缺陷的软件程序可能造成的破坏，通用型系统采用保护机制将各个软件进程从逻辑上加以隔离，其中包括环路、进程地址空间、存储保护、虚拟存储器和分离内核。为了最大限度地降低存在缺陷的硬件核可能造成的破坏，本书阐述了采用壕沟的物理隔离策略，以及如何采用吊桥技术实现核之间的受控共享。壕沟技术是通过在设计的布局阶段对各个核的布置加以限制来实现的。吊桥技术是通过对可重构设计进行静态跟踪来实现的。片上引用监视器，通过执行用专业语言表达的策略，围绕存储器，提供对核的分隔。引用监视器的设计流程，类似于电子系统层级（ESL）设计，高级策略规范可以自动地被转换为低级硬件实现。策略执行机制必须在其执行的策略的复杂程度上具有可扩展性，而且还必须在面积、循环时间以及系统整体性能上具有高效性。最后，这些方法在由一个共享 AES 核的两个域所组成的嵌入式多核的设计上进行了评估。

正确的安全性分析必须就 FPGA 如何适应更大规模的系统进行全盘考虑。本书就复杂性控制提出了几种抽象概念，从而为安全性分析提供帮助。引用监视器的概念将功能性部件与安全性部件相隔离，这些安全性部件必须体积小巧、防篡改，且不可旁路。安全架构为执行具体的策略指定了安全性部件与功能性部件的组织形式。这一策略抽象为指定系统主体和客体，以及在何种情况下某些主体能够访问某些客体的一种方法。应当使用多种互补性安全技术，因为尽管多种脆弱的安全机制组合在一起不一定能形成一种更为强大的机制，但每一种技术都有其独到的优势和局限性。安全系统工程需要在系统整个生命周期中都高度关注，其中包括需求分析、设计、实施、测试、交付、配置、操作、维护和检查等。安全方法必须可供系统设计人员和最终用户使用。

为了改进设计流程和 IP 核的安全性，今后继续研究十分必要，这些研究课题包括小型可信任工具的开发和工具与核的配置管理等。同时还需要通过研究来严格分析硬件模块的安全属性，将异常检测应用于硬件行为，对 IP 核的 HDL 规范进行形式化分析，检测并降低 FPGA 设计中隐蔽信道的风险，以及对比特流保护机制进行严格的形式化安全性分析。

很多用于改进 FPGA 系统安全性的设计技术也同样能够改进 ASIC 系统的安全性。事实上，FPGA 能够为新型硬件安全技术的原型设计提供一个理想的评估平台。由于硬件制造商通常不愿意采纳由研究人员开发的安全方法，因此 FPGA 为这些增强功能提供了一个验证平台。FPGA 目前能够承载多达八个完整的 PowerPC 软处理器核，从而为进行单芯片多处理器的安全改进实验提供了理想的环境。

与面向硬件的安全性设计有关的两个基本问题是：应当在芯片上执行什么样的策略和如何执行该策略。我们可以把核划分成若干个等价类，而安全架构则通过说明书详细规定某个核属于某等价类。策略执行机制能确保属于不同域的核不会相互干扰。拥有大量核的处理器已经应用在某些设计当中，因此以一种高效而又安全的方式进行通信管理至关重要。通信资源将主导整个芯片的面积，而将吊桥技术应用于未来的互连设计将使得多核的受控共享成为可能。

参 考 文 献

1. J. Brouchier, T. Kean, C. Marsh, D. Naccache, Temperature attacks. IEEE Secur. Priv. **7**(2), 79–82 (2009)
2. M. Gordon, Validating the PSL/Sugar specification language using automated reasoning. Form. Asp. Comput. **15**(4), 406–421 (2003)
3. J. Hannan, F. Pfenning, Compiler verification in LF, in *Proceedings of the 7th Annual IEEE Symposium on Logic in Computer Science (LICS)*, Santa Cruz, CA, June 1992
4. W.A. Hunt, Microprocessor design verification. J. Autom. Reason. **5**(4), 429–460 (1989)
5. C.E. Irvine, K. Levitt, Trusted hardware: can it be trustworthy?, in *Proceedings of the 44th Annual Design Automation Conference (DAC)*, San Diego, CA, June 2007
6. T.E. Levin, C.E. Irvine, T.V. Benzel, G. Bhaskara, P.C. Clark, T.D. Nguyen, Design princi-

ples and guidelines for security. NPS Technical Report NPS-CS-07-014, Naval Postgraduate School, Monterey, CA, 21 November 2007

7. H.G. Rice, Classes of recursively enumerable sets and their decision problems. Trans. Am. Math. Soc. **74**, 358–366 (1953)
8. H. Sasaki, A formal semantics for Verilog-VHDL simulation interoperability by abstract state machine, in *Proceedings of the Conference on Design, Automation, and Test in Europe (DATE)*, Munich, Germany, March 1999
9. D. Spinellis, Reliable identification of bounded-length viruses is NP-complete. IEEE Trans. Inf. Theory **49**(1), 280–284 (2003)
10. K. Thompson, Reflections on trusting trust. Commun. ACM **27**(8), 761–763 (1984)

附录 A　计算机体系结构的基本原理

摘要：任何探讨 FPGA 和嵌入式系统的书籍，如果不对计算机体系结构加以论述，就不能算是完整的。为了更深入地理解本书中所提出的面向硬件的安全方法，本附录对计算机结构中适用于现场可编程门阵列（FPGA）、专用集成电路（ASIC）和中央处理器（CPU）的基本概念进行了论述。

A.1　计算机架构师的日常工作是什么？

计算机架构师要将各种应用映射到物理设备上，而应用的要求将决定结构的设计。对于某些应用，通用的 CPU 架构就足以胜任，通过软件实现应用。但是，通用 CPU 有时会无法满足需求，需要定制的硬件来提供必需的性能或满足其他需求。需要定制硬件的应用的实例包括网络应用和图形应用，它们对于吞吐能力有着很高的要求。此外，嵌入式系统由于受到资源限制的先天因素，通常都需要对硬件进行定制。计算机架构师的工作就是通过权衡利弊来确定某一特定应用的最佳结构。计算机架构师的工作室配有各种工具，包括测量与度量工具、成本效益分析工具、模拟工具、被称为“基准测试程序”的标准程序，以及逻辑分析工具等。计算机架构师要进行大量的实验，并在实验中使用不同的设计参数。由于要尝试的可能的实验的数量总是大大超出可利用的时间，因此计算机架构师必须确定最重要的设计参数而把其他参数排除掉。

虽然设计定制硬件是一项艰苦的工作，但是可以在通用型系统上实现极大的性能收益。定制设计可以针对很多学科来开发，包括机器学习和神经学、生物统计学、医学[17、18]、密码学[29]、安全保护、网络工程、计算机视觉、以及程序分析等等。对于一项特定的应用，计算机架构师会通过分析来确定最通用的操作并随后对其进行优化。了解了问题的结构才有可能通过在多个硬件单元上并行进行通用操作来从应用当中提取并行性。

成为计算机架构师的那一刻是一个令人激动的时刻。用 Mark Oskin 的话来说，“七年前，当我作为一名年轻的助理教授起步时，计算机科学系的同事们都觉得计算机结构是一个已经解决的问题。像‘增大’和‘缩小’之类的词经常被用来形容这一领域内正在进行的研究……从计算机科学今后的前景考虑，应该说计算机结构曾经是一个已经解决的问题。”[19]但是，这一领域的未来存在着重新燃起的希望。曾经在过去发挥过作用的性能增强技术如今遇到了技术“难关”，包括“电源难

关”、“存储难关” 和 “指令级并行 ILP 难关”[1]。此外，大型无序式处理器设计的复杂性和可靠性也向实施和验证工作提出了挑战。业界已经采用了单芯片多处理器（CMP）技术来应对这些问题，但是开发能够通过利用多核硬件来实现性能增益的多线程软件几十年来依然是一大难题。正如美国加州大学伯克利分校实验室的研究人员所言：“业界需要研究领域提供帮助来成功实现向并行计算的近代革命性转变”。[2]明智的策略是从“我能够利用这些核做哪些用单代码机制所不能做的事情”的角度去考虑这一难题，而不是试图去平行放入现有的标量软件代码。为程序员利用多核硬件提供便利是一项公开的研究挑战[7]。除了编程问题外，还需要有切合实际的多核基准测试程序来用于评估。奥斯汀总结道，“在我的一生中，现在是计算机结构研究最令人激动的时刻；真的，年龄和资历都比我高得多的人都坚信现在是自计算机发明以来在计算机结构领域最令人激动的时刻[19]。

A.2　CPU、FPGA 与 ASIC 之间的折中方案

图 A-1 展示了 CPU、FPGA 与 ASIC 的相对通用性。CPU、FPGA 与 ASIC 之间存在着几种折中方案，包括软件对硬件、通用性对性能、成本对性能，以及通用性对安全性等。CPU 用于运行软件，对于程序而言成本相对低廉，但会产生较大开销。ASIC 属于定制硬件，拥有相对较高的性能，且成本相对昂贵。FPGA 对于程序而言比 CPU 难度更大，但比 ASIC 成本更为低廉。此外，FPGA 能够比 CPU 实现更大的处理量，但不如 ASIC 高。FPGA 与 ASIC 之间的差距正在缩小，因为 FPGA 能够采用最新的特征尺寸进行构建，经济性更好，而较低容量的 ASIC 有时采用更大的特征尺寸来制作，从而成本更为低廉。

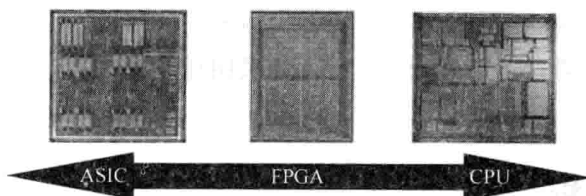


图 A-1　在通用性相当的情况下，对 CPU 进行编程最方便，而 ASIC 则相反

权衡通用性与安全性之间的折中方案比权衡其他因素之间的折中更为复杂。攻击者们会很喜欢通用性，因为他们能够在通用型系统上运行恶意软件。但是，这并不意味着 ASIC 不会有安全问题。若果真如此，那么“专用硬件”将能够解决这个世界上的所有安全问题。如果一个设备采用硬连线方式而只完成一件事情，它怎么可能受到黑客攻击呢？可悲的是，这样的思路只是一种对安全概念认识模糊的表现。专用设备对于安全社群很有用处，而限制系统的功能性可以提升安全性。但

是，系统的安全性不应当完全依赖于把一个系统的设计初衷局限于数目有限的功能上或将其设计严格保密。

通用性与安全性之间折中方案的另一个方面则涉及可信任厂家的问题。ASIC 的知识产权很容易遭到窃取和恶意入侵，因为其设计通常是发送给第三方厂家的。而在另一方面，CPU 和 FPGA 是在制作之后才进行编程的。但是，由于硬件设计可能会从现场的 FPGA 中被窃取（通过避开比特流解密机制），软件有可能从现场的通用型系统中被窃取，从而又带来了新的问题。从现场的 ASIC 中窃取设计是一种代价更为高昂的举动，需要采用反向工程攻击和打磨扫描攻击。

A.3 计算机体系结构与计算机科学

计算机体系结构跨越了计算机科学的所有领域，而且与理论、算法、语言、操作系统、网络设计、安全一样必不可少。处理器的性能增益很多都是算法发展进步的结果，而不只是在单个核上集成越来越多的晶体管的原因。晶体管排列在大规模的、正确的、有效和容错的计算机结构当中时仅能够进行计算。随着单个芯片上的晶体管数量接近甚至超过十亿，对数量如此庞大的单个部件进行统一就需要用复杂的算法来实现优化、调度、布局、布线、验证、并行、并发、多线程、超线程、多重处理、推理、程序概要分析、一致性、乱序执行等。了解处理器是如何被设计的以及现代处理器的内部有哪些构造是计算机体系结构的基本问题。

处理器是人类所构造的最为复杂的事物。其复杂性是造成芯片制造公司中的大部分工程师是验证工程师，而只有一小部分才是设计工程师的原因。随着芯片制造业追求越来越小的特征尺寸，因而使得非重复性工程的成本日益增长，芯片设计错误即便对于一家大公司也是致命的。由于一项由十亿个晶体管构成的设计，其可能的状态数目为天文数字，因此验证工作需要采用高级算法，而且其难度与设计的复杂性是成比例的。

A.4 程序分析

计算机体系结构研究的是“什么能够使得计算机运行得如此之快”的问题。为了编写出快速的程序，必须要了解处理器内部在进行着哪些工作。在现代处理器内部，每一秒钟要发生数量庞大的事件，大约有 10 亿次，而要实时分析这些事件无疑是一件浩大的工作。

A.4.1 处理器仿真科学

仿真是一项基本的程序分析技术，而处理器仿真则囊括了整个研究领域。计算

机架构师花费大量的时间和处理器周期来运行仿真程序。仿真不仅有助于分析处理器在运行程序时的性能，而且对处理器进行仿真对于检验其正确性也是必需的。正如一名建筑师在破土动工之前必须绘制蓝图一样，在制造芯片之前也必须对处理器进行仿真。在仿真和制造过程之间，通常还要在 FPGA 上进行原型样机的设计以及对设计进行测试。设计的仿真应当始终在 FPGA 实现之前（因为在 FPGA 上进行原型样机设计需要付出极大努力）和芯片制造之前（因为高额的非重复性工程制造成本）进行。

仿真技术的选择取决于需要解决的问题。在精细的粒度级别上进行的仿真在计算上为高度密集型。例如，在周期级仿真当中，处于单时钟周期颗粒度上的事件会被运行在不同硬件环境下的软件程序所复写，因而该仿真非常详细，从而在计算上也十分密集。在指令级仿真（也称为模拟）当中，处在指令颗粒度上的事件在另一环境处理下得到表现，因此该仿真比周期级仿真成本更低，但精确度也较低。模拟对于提炼好的想法很有用，因为它快捷而简明，但对于更深入的分析则用处不大，因为它不够精确。例如，模拟对于为某个特定的基准程序确定存储访问的数量和缓存的未命中数量会很有帮助。包括周期级和指令级在内的所有模拟器，其运行速度都要比其所模拟的处理器的速度低很多。

关键的仿真参数是仿真的时间长度和内容。由于要想仿真出长时间运行程序的所有事件往往是不切实际的，因此必须选择一份能够代表该程序的最关键行为的样本。从该程序的最开头来采集这样的样本是一个不好的选择，因为计算机程序一般是通过将数组归零来开始的。关键性的动作通常都发生在程序的中间段。在指令级仿真中，往往要采集包含大约 1 亿条指令的样本。而对于周期级仿真，所需的样本则更小。另一个关键的仿真参数是实验所使用的基准测试程序。基准测试程序是用来比较处理器的标准程序。该程序是由正在被仿真的处理器来运行的。在描述实验方法时要将各种仿真参数整理成文档以便了解各个结果的含义，这一点极其重要。

为研究对正在被仿真的处理器各个方面加以改变后的效果，计算机架构师会修改仿真器的源代码以便能够观察到新的行为。实现这一观察目的的一种方式是生成跟踪文件，然后在稍后的离线状态下对其进行分析，这一方法被称为“跟踪驱动型”仿真。在跟踪驱动型仿真当中，会对仿真器进行修改，以便将特定的事件写入到“跟踪文件”中。跟踪文件可能会变得非常大，程序每执行 1s，其大小约为数千兆字节，而向跟踪文件中写入则会进一步减慢仿真过程。在跟踪文件生成之后，计算机架构师会使用另一个程序来对该跟踪文件进行离线分析。例如，该程序可能会为一项使用不同的存储管理策略的仿真工作来确定最佳缓存替换策略。或者，该程序也可能会通过一系列分支事件的仿真过程来确定其最佳分支预测技术。但是，由于跟踪驱动型仿真是静态的，并不适合于研究涉及重大推理性执行过程的行为，因此它的应用范围受到了一定限制[30]。

SimpleScalar 是一个供编译使用的单处理器仿真器和工具套装[4]。该套装包含有 SIM-FAST（一个不提供时序的功能仿真器）、SIM-OUTORDER（一个具备详细时序模式的功能与时序仿真器）、SIM-CACHESIM（用于仿真存储行为）、以及 SIM-BPRED（用于仿真分支预测）。SimpleScalar 能够运行具有不同指令集架构（ISA）的程序，其中 MACHINE. DEF 文件可以指定不同的 ISA，包括 Pisa、Alpha、Arm 或 x86。Pisa 是一套基于 MIPS 的架构，某一特定版本的 gcc 可以生成 Pisa 代码。Alpha 支持真正的编译程序和一组二进制文件。Arm 主要用于嵌入式系统中，而 x86 则可以通用。SIM-MAIN 是主要的仿真器环路。

A. 4. 2 片上分析引擎

仿真通常会使程序的执行减慢约一个数量级，因此使用片上分析硬件捕获并分析实时事件对于计算机结构的研究十分有益。片上分析模块能够以高带宽捕获和分析实时事件，而无需将每个事件记录到硬盘上的跟踪文件中。虽然某些处理器具备分析功能，比如拥有专用的分析寄存器等，但由于经济上的原因，很少见到计算机结构研究人员能拥有功能较全面的分析引擎。芯片制造商们忙于验证处理器设计本身，而无暇顾及分析模块的设计和验证，大多数最终用户由于不是计算机结构研究人员，所以从未使用过片上分析模块。

为了克服这一问题，一个办法是采用协同处理器来卸载计算密集型分析任务。协同处理器可以是与主处理器安装在同一电路板上的一块芯片，也可以是与主板的 PCI 接口相连的一块 FPGA 板卡，如图 A-2 所示。当处理器执行程序时，软件测试仪器会采集分析数据，并把这些数据存入缓冲区。随后将缓冲区中的分析数据写入 PCI 驱动程序，而不是在主 CPU 上的软件中对其进行计算密集型分析。接下来，FPGA 分析模块会处理这些分析数据，其输出结果供优化程序、操作人员、显示器、或远程监控装置使用。显然，这一联机程序分析结构比片上分析模块的带宽要低。实现更高带宽的方法是采用 3-D 堆迭集成技术。这种技术，可把多个独立制造的硅芯堆迭起来，制成 3 维集成电路，其性能大大增强，有很大的商业价值[15,16]。该 3-D 堆迭集成电路中包含一个分析模块，用于该商业芯片上的事件分析。将 3-D 堆迭集成技术应用到安全领域中也得到推荐[14]。采用该方法，在 3-D 集成电路的堆迭层中有一层硅片被称为“3-D 控制平面”，其电路功能是实现专用的安全函数，该层负责监控并执行被称为“计算平面”层的安全策略。

A. 4. 3 二进制测试设备

二进制测试设备是另一项有用的程序分析技术[12]，可以独立使用，也可以作为片上分析引擎的一个部件，如图 A-2 所示。通过对自定义函数的调用，二进制测试设备能响应特定事件，记录下原始的未修改的二进制数据流。这些软件函数能够

对事件进行分析，或仅仅将事件的某些特定细节写入跟踪文件。与仿真类似，该测试过程的运行速度至少降低约一个数量级，若需要将测试结果写入跟踪文件会使测试过程变得更慢。但是，二进制测试设备为研究复杂的多线程应用程序（例如 web 浏览器、文字处理软件以及图形编辑程序等的行为）提供了机会。这些应用程序的具体功能能够通过与用户界面的交互而被调用（如调用菜单命令、对话框等）。

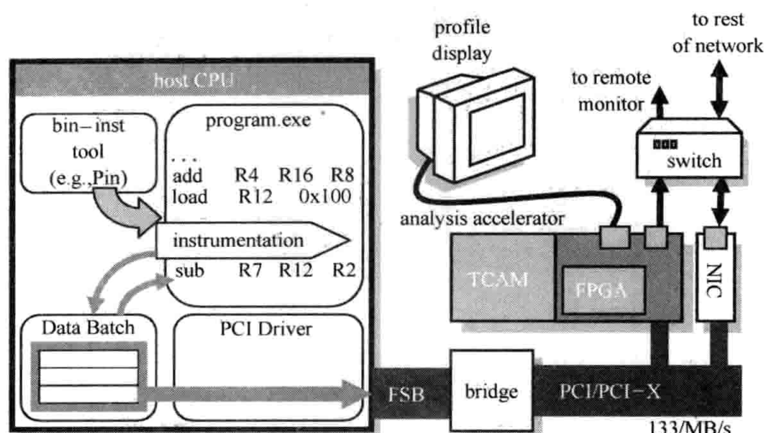


图 A-2　联机程序分析结构。计算密集型程序分析任务能够被卸载到 FPGA 协同处理器中

A. 4. 4　相位分类

相位分析是计算机架构师工作室中另一个有用的工具[21,22]。计算机程序在其执行过程中会显示出重复行为。识别相位，即确认相似行为之间的时间间隔，可以为指导运行优化和缩短仿真时间提供帮助[8,9,20,23]。相位分类是通过统计每个基本块㊀在一个时间间隔内被执行的次数来工作的。基本块矢量（BBV）只是一个数组，该数组中的每个项用于存储每个基本块的执行次数，该项的权重取决于每个基本块中的指令个数，并根据间隔期间被执行的基本块的总数进行校正（做归一化处理）。度量相似性的量纲被称为“曼哈顿距离”。利用“曼哈顿距离”可表示不同 BBV 之间的相似程度，它是两个 BBV 的每个元素之间差值的绝对值之和。随机投影被用来减少数据的维度，而 k-均值聚类法被用来将 BBV 归入不同的集群。一个集群中的所有 BBV 都属于同一相位。相位分类的其他结构还包括捕捉存储访问步幅的结构、使用工作集概念的结构，甚至还有使用小波系数的结构[13]。图 A-3 显示了 Firefox 在加载网页时处理 5000 万条指令的存储行为，其中根据基于小波的

㊀ 一个基块是一段顺序执行的代码，只有一个入口和一个出口，没有跳转指令。

相位分类算法为间隔区进行了着色。

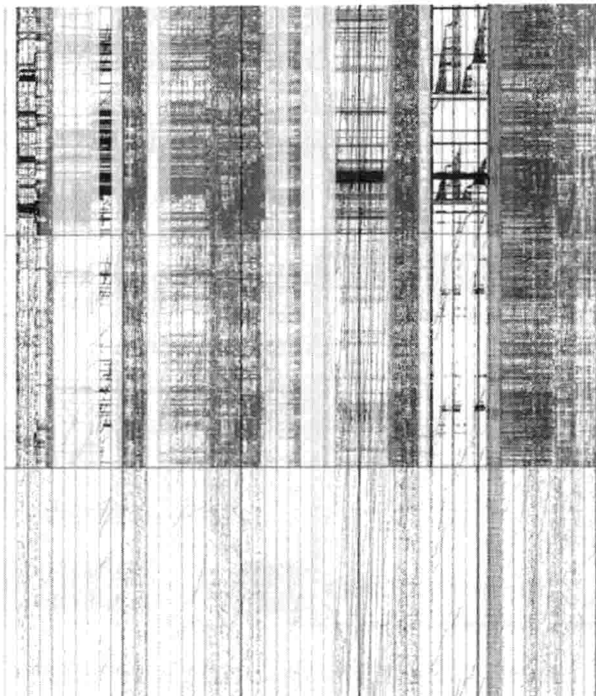


图 A-3　Firefox 在加载网页时处理 5000 万条指令的一块缓存区域。x 轴为时间，y 轴为存储访问地址的一个函数。每一条“垂直线”或间隔区代表 100 万条指令。顶部区段显示的是一级缓存的命中次数，中部区段显示的是一级缓存的未命中次数，而底部区段显示的是二级缓存的未命中次数。根据基于微波的相位分类算法为间隔区进行了着色[13]

A.5　新型计算机架构

充分而有效地利用单个芯片上集成的数十亿个晶体管和大量的核是下一代处理器设计方案的目标。成功的关键是通过使用稳健而不隐藏技术细节的设计理念来控制复杂性。计算过程的成本将更加低廉，但是通信成本会更加昂贵。管理大量核之间的通信的互连网络将有可能消耗大部分的片上资源。正如处理器使用分级存储体系那样，未来的处理器有可能会使用分级互连体系。

A.5.1　DIVA 结构

数据增强结构（DIVA）[3] 是一种尝试管理复杂性的技术。采用预测执行机制的处理器验证工作在计算上属于密集型。其关键理念是，从计算角度而言，计算结

果的正确性要求低于该结果的计算过程。因此，采用乱序执行机制的复杂处理器的运行时间的正确性能够由具有“检测功能”的较小的硬件模块来进行验证。该检测单元的小型体积使其更易于验证。该检测单元位于芯片上，与采用预测机制的更为复杂的处理器部署在一起。DIVA 概念与引用监视器概念之间存在着几个相似之处。两者都很小巧，从而更容易实现检验功能；两者都有助于对复杂性进行控制，且两者都属于部署在芯片上的运行机制。

A.5.2 原生微处理器

为了控制复杂性，未来的处理器有可能采用大量的计算核。这些核将以更高效的方式互相沟通，因此需要有新的方案替代传统的总线互连技术。从以计算为核心的设计到以通信为核心的设计的转变正在进行当中。片上网络（NoC）的理念早已被提出[6]，但因其复杂性而很少实现。片上网络的实施需要建立部署在整个芯片上的微型网络路由器。该芯片被分为若干个板块，每个板块拥有一个计算核、一个网络接口（NI）和一个交换机。各种拓扑结构都已做了研究，例如分级式环形互连结构[5]。不过目前仍然缺乏实用的基准评估程序、网络流量模型和仿真环境。由于 NoC 的主要应用领域是嵌入式系统，因此 Intel 公司提出，未来互连网络技术的发展必须为片上互连网络制定一个简化的微型网络协议堆栈[11]。美国麻省理工学院开发的原生微处理器结构，同时采用了静态和动态路由技术[25,28]。Tilera 公司已经开发出了一种针对嵌入式系统市场的 64 核的处理器。Tilera 开发的处理器名为 TILE64，该处理器基于 MIT 原生结构。

A.5.3 WaveScalar 结构

WaveScalar[24]是一种数据流结构，它是冯·诺依曼结构的备选结构。冯·诺依曼结构的问题在于，所有数据，包括驻留在分级存储体系（如片外 DRAM）低级别上的数据，都必须传送到中心位置（CPU）才能处理，然后再发回内存。WaveScalar 结构的关键理念是在存储系统内部的适当位置上来执行程序。变量值发生改变时，会自动重新计算依靠该变量的其他变量的值。WaveScalar 利用了“局部性原理”，即如果程序访问位于位置 i 上的某个内存值，那么该程序很有可能会在不久之后访问与 i 相邻的某个位置。WaveScalar 处理器的基本构造块是 WaveCache，它由存储部件和处理部件共同组成。

A.5.4 应用于医学领域的结构

计算机体系结构在医学领域有着广泛的应用。例如，医疗设备中的亚阈值电压处理器可被用来防止失明[17,18]。亚阈值电压处理器用处理器性能来换取能源节约。该设备使用亚阈值电压处理器来测量眼压，从而延缓青光眼和糖尿病患者失明症状

的发作。

“该系统的设计目的是将眼球的内表面（玻璃体）绷紧。该系统将通过门诊科来安装，并为患者提供对眼内压的实时反馈。最近的医疗研究表明，严密的监护和随后的眼压控制能够延缓青光眼和糖尿病患者失明症状的发作。该眼压测量系统包括一颗亚阈值传感处理器、384 字节内存、1024 字节 ROM，一个基于 MEMOS 的压力传感器、一种利用眼球内部温度梯度来发电的基于珀尔帖效应的能量采集机制、以及一套基于电感耦合的通信系统[18]。

斯坦福大学的一个研究小组正在进行视网膜假体植入技术的攻关。该植入假体被放置在视网膜后面，拥有一组微型太阳电池，而其他小组则通过射频信号向他们的设备传送电能[10]。

A.6 存储器

计算机分级存储体系可以比喻成一个厨房。当厨师需要一份配料时，第一个要查看的地方是电冰箱。电冰箱就好比是高速缓存。能够找到所需的配料就好比是缓存命中，而没有找到就相当于缓存没有命中。如果该配料不在电冰箱里，就必需查看食品储藏室。这个食品储藏室就好比是二级缓存。如果所需的配料也不在食品储藏室，那就好比是二级缓存也没有命中，那么就需要去超市。由于访问片外存储器所需时钟周期的数量相对比较庞大，因此超市就好比是片外存储器。从食品储藏室拿一样物品所花的累计时间比从厨房电冰箱里拿所花的时间要长，而开车去超市所花的累计时间比从食品储藏室拿配料所花的时间也要长。

存储器被分组排列，如图 A-4 所示。一个存储地址必须指定分组以及该组内的既定位置的行与列。行解码器和列解码器会选择该地址中所指定的行与列。该比特位就存储在指定的行和列的交叉点上。图 A-5 显示了一个比特位存储到 SRAM 单元内的过程。其存储机制是达到稳定平衡的两个非门之间的反馈。图 A-6 显示了另一组稳定平衡。每个 SRAM 单元需要六只晶体管：每个非门分配两只，最后还有两只额外晶体管。图 A-7 展示的是一个允许多次同时访问的多端口 SRAM 存储单元，这对于同时进行读取和写入或者向同一位置进行多次访问的视频卡非常有用。

尽管对功耗和面积有着极大的需求，SRAM 仍然被用于片上一级缓存和寄存器，因为这两者都需要高性能。另一方面，片外 DRAM 的每个比特位所使用的晶体管数量更少，从而减少了每个比特位的功耗和面积。这使得非常密集地部署多个存储器成为可能，但同时也会降低性能（SRAM 的操作周期为 1ns，而 DRAM 的操作周期为 100ns）。DRAM 存储单元使用的是电容器，该电容器如同一个电子“桶”。向 DRAM 中写入要涉及到充电或放电，而读取则更为麻烦。由于电容器在缓慢放电，其值为“1”的电子桶需要被重新充电。过程中所涉及的电流很细微，

100 电子就代表了一个比特位。图 A-8 显示的是一个 DRAM 存储单元，其中电容器画在了图中右下角的位置。

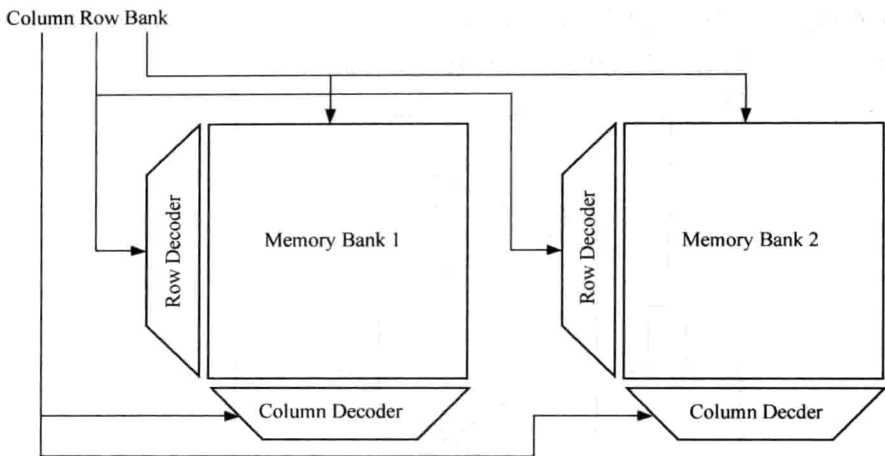


图 A-4 存储器分组排列，并配有行解码器和列解码器

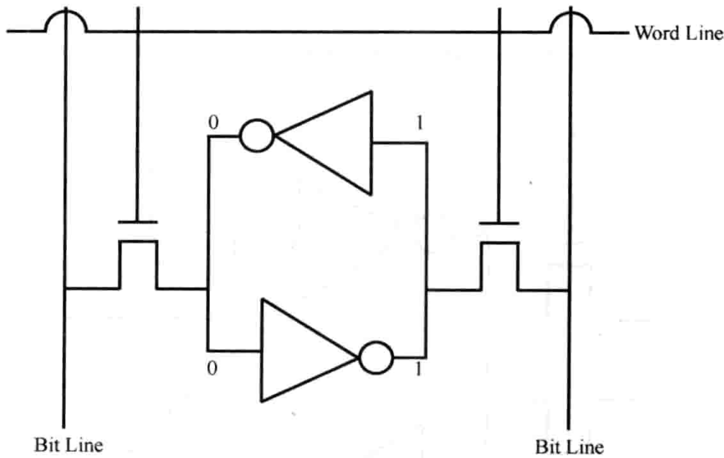


图 A-5 一个 SRAM 存储单元存储一个比特位。其存储机制是达到稳定平衡的两个非门之间的反馈

由于处理器与存储器之间的性能差距在不断拉大，系统设计人员采用了一种分层存储结构，即由冯·诺依曼于 1946 年提出的一种概念。为了捕获空间局部性，缓存被分解成若干个区块。在一个“全相联”的缓存中，一个区块可以到达该缓存内的任何位置。这就需要建立一种标记体系以便知道哪些存储区块驻留在缓存内，同时还需要并行搜索所有需要的插槽。处于另一个极端的是“直接映射”缓存，该类缓存的工作方式如同一张散列表，因为一个区块只能到达一个位置。直接

映射缓存的速度非常快，但有可能与同一插槽发生地址冲突。处于上述两个极端之间的是“组相联”缓存，该类缓存允许一个区块到达一个集合内的各个可能的地址。缓存替换策略包括最近最少使用（LRU）原则、最低使用频率（LFU）原则、随机和“预测”原则，该原则会对即将使用的存储区块进行预测。

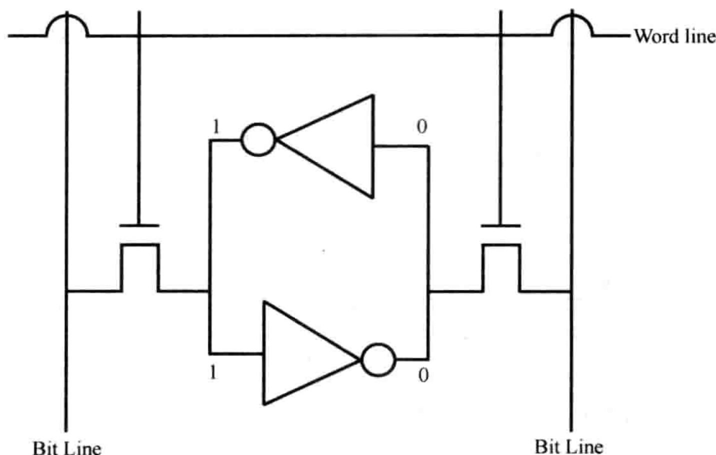


图 A-6　SRAM 存储单元的另一组稳定平衡

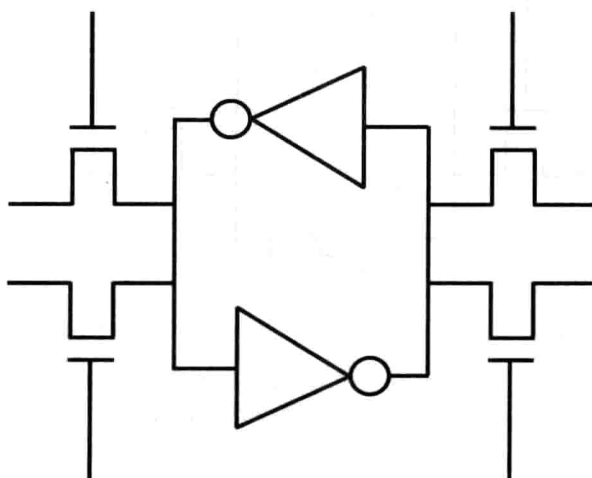


图 A-7　允许多次同时访问的多端口 SRAM 单元

公用程序优化技术是改变程序访问存储器的方式，以降低缓存的未命中次数。该程序经过重新设计，使得所需的数据能更好地与缓存相吻合，并能更有效地被预取。

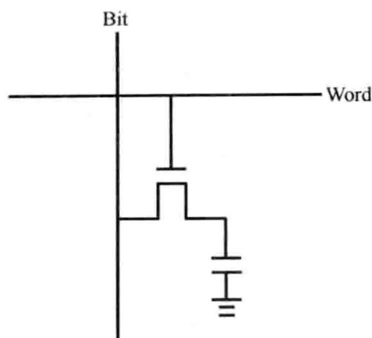


图 A-8　DRAM 存储单元

A.7　超标量处理器

对代码相关性的分析可以确定何时需要更多资源（如加法器）来并行执行多条指令。多发问题处理器包括“超长指令字（VLIW）”计算机和超标量计算机。在一台超长指令字（VLIW）计算机中，两条指令能够被“黏合”在一起而形成一个长指令，并由编译程序来进行调度。在超标量计算机中，由处理器硬件来安排各条指令。目前，大部分工作都是由硬件来完成的，而不是编译程序。例如，硬件对指令进行动态整理的“乱序执行”机制目前正在普及。

Tomasulo 算法是一种用于发现并行性的分布式可扩展算法[26]。针对数据的相关性，乱序执行机制会动态地适应数据流图。在某种意义上，数据流图是通过一根“吸管”被“吸取”的，而且由于芯片上存在着有限的资源，因而有时会出现未命中的情况。Tomasulo 算法采用动态循环展开、混合与匹配，以及保留站。指令会在保留站等候，直至获得所需。在执行之后，它们已获得所需的事实会通过广播总线进行广播。Tomasulo 算法在 IBM360/91 上得到了实施，尽管那在商业上是一次失败的实践。它拥有四个浮点寄存器，但得到的编译器支持却微乎其微。在进行加法运算时，指令通过检查寄存器文件等待它所需要的两个操作数。其计算结果会通过公用数据总线在各处进行广播。保留站方案需要提供操作（如加法、减法）的入口、操作数、起点的保留站名称以及保留站是否被占用。

Tomasulo 算法的问题包括通过公用数据总线向所有人进行广播。由于次序举足轻重，因此需要仲裁来为通信确定优先次序。由于总线过载是一大难题，因此可能需要多个总线，或者需要点对点的“交叉开关”实现网络的互连。复杂性是 Tomasulo 算法的另外一大难题。Tomasulo 算法告诉我们：优秀的解决方案应当是一种“折中方案”。Tomasulo 算法的一种备选调度方案是由奔腾 4 所采用的“适时调度”。

针对名称的相关性，执行 Tomasulo 算法的硬件采用了寄存器重命名机制。虽然用户和编译器只能看到数量较少的结构寄存器（如 32 个），但是物理寄存器的集合可以更大（如 256 个），而寄存器重命名机制会将较小的集合映射到较大的集合上。寄存器重命名机制加大了 Tomasulo 算法设计和验证的复杂性。

A.8 多线程

图 A-9 展示了在超标量处理器上执行的两个线程，并说明了“垂直浪费”的问题，即多余的硬件资源经常得不到利用[27]。当处理器在一个周期内未发布指令时，Tullsen 等人就将空发射槽定义为垂直浪费。而当所有可用的发射槽在一个周期内均未被填满时，就会出现水平浪费[27]。在图 A-9 当中的一个点上，线程 2 无法使用任何硬件资源。为了缓解这一问题，一种被称为“细颗粒度多线程”的方案试图将两个线程交错，但是这不会消除对资源的“水平浪费”，如图 A-10 所示。

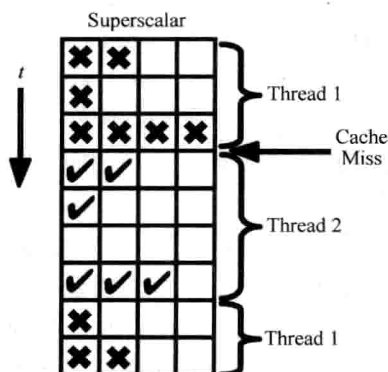


图 A-9 超标量计算机展示的垂直浪费

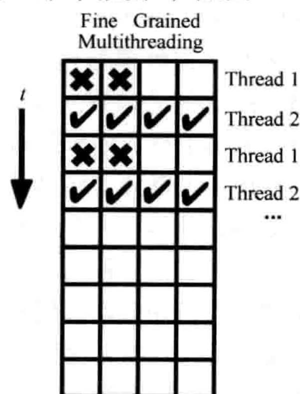


图 A-10 细粒度多线程计算机展示的水平浪费

虽然交错机制解决了垂直浪费的问题，但是依然存在所有硬件资源未被充分利用的情况。同步多线程计算机每个周期都将指令交错并同时执行，如图 A-11 所示。同步多线程计算机出色地完成了对指令的“打包”工作。同步多线程计算机的技术问题包括以下内容：

1）页面表和 TLB（TLB 是页面表的缓存）；
2）寄存器的赋值（需要独立的寄存器）；
3）多个堆栈；
4）分级存储体系（一个线程可能会重写缓存）；
5）单独的分支预测器；

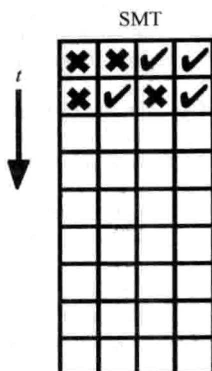


图 A-11　同步多线程（SMT）计算机每个周期都将指令进行交错，从而将其同步执行。该方法出色地完成了对指令的“打包”工作，并且更加充分地利用了多余的硬件资源

6）并行的指令发布逻辑；

7）多个线程之间共用一个结构。

参考文献

1. K. Asanovic, R. Bodik, B.C. Catanzaro, J.J. Gebis, P. Husbands, K. Keutzer, D.A. Patterson, W.L. Plishker, J. Shalf, S.W. Williams, K.A. Yelick, The landscape of parallel computing research: a view from Berkeley. Technical Report No. UCB/EECS-2006-183, University of California, Berkeley, 18 December 2006
2. K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, K. Yelick, A view of the parallel computing landscape. Commun. ACM **52**(10), 56–67 (2009)
3. T.M. Austin, DIVA: a reliable substrate for deep submicron microarchitecture design, in *Proceedings of the 32nd International Symposium on Microarchitecture (MICRO-32)*, Haifa, Israel, November 1999
4. T. Austin, E. Larson, D. Ernst, SimpleScalar: an infrastructure for computer system modeling. IEEE Comput. **35**(2), 59–67 (2002)
5. S. Bourduas, Modeling, evaluation, and implementation of ring-based interconnects for network-on-chip. Ph.D. dissertation, McGill University, Dept. of Electrical and Computer Engineering, Montreal, Canada, May 2008
6. W.J. Dally, B. Towles, Route packets, not wires: on-chip interconnection networks, in *Proceedings of the 37th Design Automation Conference (DAC)*, Las Vegas, NV, June 2001
7. J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
8. L. Eeckhout, R.H. Bell Jr., B. Stougie, K. De Bosschere, L.K. John, Control flow modeling in statistical simulation for accurate and efficient processor design studies, in *Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA)*, Munich, Germany, June 2004
9. L. Eeckhout, J. Sampson, B. Calder, Exploiting program microarchitecture independent characteristics and phase behavior for reduced benchmark suite simulation, in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'05)*, Austin, TX, October 2005

10. W.D. Jones, A form-fitting photovoltaic artificial retina. *IEEE Spectrum*, **46**(12), December 2009. http://spectrum.ieee.org/biomedical/bionics/a-formfitting-photovoltaic-artificial-retina
11. D. Kanter, The Common System Interface: Intel's future interconnect. *White Paper, Real World Technologies*, August 2007
12. C. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V.J. Reddi, K. Hazelwood, Pin: building customized program analysis tools with dynamic instrumentation, in *Proceedings of the 2005 ACM/SIGPLAN Conference on Programming Language Design and Implementation*, Chicago, IL, June 2005
13. T. Huffmire, T. Sherwood, Wavelet-based phase classification, in *Proceedings of the Fifteenth International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Seattle, WA, September 2006
14. T. Huffmire, J. Valamehr, T. Sherwood, R. Kastner, T. Levin, T.D. Nguyen, T. Sherwood, Trustworthy system security through 3-D integrated hardware, in *Proceedings of the 2008 IEEE International Workshop on Hardware-Oriented Security and Trust (HOST)*, Anaheim, CA, June 2008
15. S. Mysore, B. Agrawal, S.-C. Lin, N. Srivastava, K. Banerjee, T. Sherwood, Introspective 3D chips, in *Proceedings of the Twelfth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, San Jose, CA, October 2006
16. S. Mysore, B. Agrawal, S.-C. Lin, N. Srivastava, K. Banerjee, T. Sherwood, 3-D integration for introspection, in *IEEE Micro: Micro's Top Picks from Computer Architecture Conferences*, January–February 2007
17. L. Nazhandali, B. Zhai, J. Olson, A. Reeves, M. Minuth, R. Helfand, S. Pant, T. Austin, D. Blaauw, Energy optimization of subthreshold-voltage sensor network processors, in *Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA)*, Madison, WI, June 2005
18. L. Nazhandali, M. Minuth, B. Zhai, J. Olson, T. Austin, D. Blaauw, A second-generation sensor network processor with application-driven memory optimizations and out-of-order execution, in *Proceedings of the International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES)*, San Francisco, CA, September 2005
19. M. Oskin, The revolution inside the box. Commun. ACM **51**(7), 70–78 (2008)
20. E. Perelman, G. Hamerly, M. Van Biesbrouck, T. Sherwood, B. Calder, Using SimPoint for accurate and efficient simulation, in *International Conference on Measurement and Modeling of Computer Systems*, San Diego, CA, June 2003
21. T. Sherwood, E. Perelman, G. Hamerly, B. Calder, Automatically characterizing large scale program behavior, in *Proceedings of the Tenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, San Jose, CA, October 2002
22. T. Sherwood, E. Perelman, G. Hamerly, S. Sair, B. Calder, Discovering and exploiting program phases, in *IEEE Micro: Micro's Top Picks from Computer Architecture Conferences*, November–December 2003
23. R. Srinivasan, J. Cook, S. Cooper, Fast, accurate microarchitecture simulation using statistical phase detection, in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'05)*, Austin, TX, March 2005
24. S. Swanson, K. Michelson, A. Schwerin, M. Oskin, WaveScalar, in *Proceedings of the 36th International Symposium on Microarchitecture (MICRO)*, San Diego, CA, December 2003
25. M.B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, A. Agarwal, The RAW microprocessor: a computational fabric for software circuits and general-purpose programs. IEEE Micro **22**(2), 25–35 (2002)
26. R.M. Tomasulo, An efficient algorithm for exploiting multiple arithmetic units. IBM J. Res. Develop. **11**(1), 25 (1967)
27. D.M. Tullsen, S.J. Eggers, H.M. Levy, Simultaneous multithreading: maximizing on-chip parallelism, in *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, Santa Margherita Ligure, Italy, 1995

28. E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe, A. Agarwal, Baring it all to software: RAW machines. IEEE Comput. **30**(9), 86–93 (1997)
29. L. Wu, C. Weaver, T. Austin, CryptoManiac: a fast flexible architecture for secure communication, in *Proceedings of the 28th Annual International Symposium on Computer Architecture (ISCA)*, Gothenburg, Sweden, July 2001
30. C. Zilles, G. Sohi, Master/slave speculative parallelization, in *Proceedings of the 35th International Symposium on Microarchitecture (MICRO)*, Istanbul, Turkey, November 2002

国际视野 科技前沿

国际信息工程先进技术译丛

《FPGA安全性设计指南》
《自主式传感器系统的能量收集——设计、分析以及实践应用》
《基于视觉的自主机器人导航》
《无线神经接口的超低功耗集成电路设计》
《基于片上去耦电容的配电网络》（原书第2版）
《智能摄像机》
《车载系统和安全的数字信号处理》
《嵌入式系统设计 —— 嵌入式信息物理系统基础》（原书第2版）
《纳米封装 —— 纳米技术与电子封装》
《内容分发网络》
《全面的功能验证：完整的工业流程》
《无线Mesh网络架构与协议》
《UMTS蜂窝系统的QoS与QoE管理》
《半导体制造与过程控制基础》
《WCDWA原理与开发设计》
《下一代移动系统:3G/B3G》
《IMS:IP多媒体概念和服务》（原书第2版）
《下一代无线系统与网络》
《深入浅出UMTS无线网络建模、规划与自动优化：理论与实践》
《HSDPA/HSUPA技术与系统设计——第三代移动
《通信系统宽带无线接入》
《无线传感器及元器件:网络、设计与应用》
《印制电路板——设计、制造、装配与测试》
《IPTV与网络视频：拓展广播电视的应用范围》
《多电压CMOS电路设计》
《微电子技术原理、设计与应用》
《蜂窝网络高级规划与优化2G/2.5G/3G/…向4G的演进》
《基于蜂窝系统的IMS——融合电信领域的VoIP演进》
《无线网络中的合作原理与应用》
《电生理学方法与仪器入门》
《移动电视：DVB-H、DMB、3G系统和富媒体应用》
《环境网络：支持下一代无线业务的多域协同网络》
《基于射频工程的UMTS空中接口设计与网络运行》
《未来UMTS的体系结构与业务平台：全IP的3G CDMA网络》
《UMTS-HSDPA系统的TCP性能》
《宽带无线通信中的空时编码》
《数字图像处理》（原书第4版）



上架指导 工业技术 / EDA和FPGA设计

ISBN 978-7-111-45783-1 定价：58.00元